FACULDADE DE CIÊNCIAS SOCIAIS E HUMANAS
UNIVERSIDADE NOVA DE LISBOA

Programação Web
UX/UI Design

Fátima Carolina França Freitas

Relatório de Estágio de Mestrado em Novos Media e Práticas Web

Março, 2017

Relatório de Estágio apresentado para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Novos Media e Práticas Web realizado sob a orientação científica do Professor António Câmara.

AGRADECIMENTOS

Um obrigada aos meus pais, família e amigos por todo o apoio e incentivo prestados ao longo de todo o meu percurso académico.

Um obrigada à FCSH-UNL por me oferecer a oportunidade de realizar um estágio para fins de conclusão de mestrado, ajudando-me a criar uma ligação entre o mundo académico e o mundo profissional.

Um obrigada ao meu orientador, Professor António Câmara, pela disponibilidade demonstrada e pelos conselhos que me ajudaram na realização deste relatório.

E, por fim, um obrigada às equipas Mobiag e Logistema que me receberam com grande hospitalidade e proporcionaram um ótimo ambiente de trabalho, tornando todo o percurso mais fácil de superar, fazendo-me sentir integrada e motivada.

PROGRAMAÇÃO WEB

UX/UI DESIGN

FÁTIMA CAROLINA FRANÇA FREITAS

RESUMO

PALAVRAS-CHAVE: Aplicação Mobile, Usabilidade, Design de experiências, Design de Interfaces, Documentação, API, Markdown, XML, HTML, CSS, REST, SOAP, Serviços Web.

O estágio realizado na Mobiag, empresa de desenvolvimento de software de carsharing, para fins de conclusão do Mestrado em Novos Media e Práticas Web, realizou-se de 10 de Outubro de 2016 a 10 de Janeiro de 2017 e dividiu-se em dois projectos: na avaliação da usabilidade da aplicação mobile Citydrive, um dos clientes da respetiva empresa, e na gestão e criação de conteúdo para a documentação da API da Mobiag.

Este relatório irá abordar as estratégias utilizadas para fazer a avaliação da usabilidade da aplicação mobile e a explicação das novas tecnologias e ferramentas aprendidas e utilizadas para poder trabalhar na documentação da API.

ABSTRACT

KEYWORDS: Mobile App, Usability, UX Design, UI Design, Documentation, API, Markdown, XML, HTML, CSS, REST, SOAP, *Web Services*.

The internship at Mobiag, a carsharing software development company, to conclude my Master's Degree in Novos Media e Práticas Web, started at October 10th 2016 and ended at January 10th 2017 and it was divided into two projects: usability evaluation of Citydrive mobile app, one of the company's clients, and management and content creation for the API documentation.

This report will approach the required strategies to evaluate the mobile app usability and will also explain the new technologies and tools learned and required to be able to work in the API documentation.

ÍNDICE

Introdução 1

Parte I – Contextualização 3

Capítulo 1: Caracterização da Empresa 4

1.1. Mobiag 4

1.2. Colaboradores 4

1.3. Produtos 5

1.4. Clientes 5

1.5. Conceito *carsharing* 5

Capítulo 2: Enquadramento Teórico 7

2.1. Definição de conceitos – Projeto 1 7

2.1.1 Conceito de Usabilidade 7

2.1.2 Usabilidade em dispositivos móveis 9

2.1.3 Design de Experiências *(UX Design)* 10

2.1.4 Técnicas de avaliação da usabilidade 11

2.1.5 Design de Interfaces *(UI Design)* 14

2.2. Definição de conceitos – Projeto 2 15

2.2.1 O que é uma API 15

2.2.2 O que é HTTP 16

2.2.3 O que é XML 16

2.2.4 *Web Services*, SOAP e REST 17

2.2.5 Sintaxe Markdown 18

Parte II – Projetos 19

Capítulo 3: Citydrive – Avaliação da Usabilidade 20

3. 1. O que é a Citydrive e como funciona 20

3. 2. Estudo da aplicação Citydrive 21

3. 3. Análise da concorrência: *Car2Go* e *DriveNow* 26

Capítulo 4: Documentação da API 31

4. 1. Instalação e acesso remoto 31

4. 2. Introdução à API Mobiag 32

4. 3. Familiarização com as linguagens e programas 32

4. 4. Organização e edição do código 37

Conclusão 39

Bibliografia/Webgrafia 42

Glossário 45

Anexos – Capítulo 3 46

Anexos – Capítulo 4 68

INTRODUÇÃO

Sendo licenciada em Design de Media Interativos, sempre tive mais presente uma componente prática do que teórica ao longo do meu percurso académico, o que me levou a escolher este mestrado e a componente não letiva de estágio com relatório. Considero que o estágio não só é a maneira mais eficaz de aplicar e solidificar os conceitos adquiridos ao longo da licenciatura e mestrado como também é uma ótima oportunidade de ganhar experiência no mercado do trabalho o que, hoje em dia, é imprescindível e difícil de conseguir.

Com isto em mente, pesquisei e candidatei-me a várias empresas com que me identificava e a Mobiag, após uma entrevista, disponibilizou-se de imediato a receber-me como sua estagiária. Desta maneira, iniciei o meu estágio curricular para fins de conclusão de mestrado, nesta empresa de desenvolvimento de software de carsharing durante 3 meses, a 10 de Outubro de 2016 e terminei a 10 de Janeiro de 2017. Durante estes 3 meses fui responsável por fazer a avaliação do design e da usabilidade da aplicação Citydrive, um dos clientes da Mobiag. Fui também responsável pelo teste do código presente na documentação da API da Mobiag como também pela gestão e criação de conteúdo para a mesma.

O presente relatório está estruturado em duas partes, com dois capítulos cada, que irão mostrar detalhadamente os métodos que utilizei para realizar estes dois projetos.

O primeiro capítulo, contextualiza o leitor, no sentido em que fala um pouco sobre a Mobiag, os seus clientes, produtos, colaboradores e cultura empresarial. Posteriormente, isto será útil para ajudar a compreender as funcionalidades da app Citydrive, mostrando o bom e o mau da mesma e irá também realçar a importância de uma boa documentação da API.

O segundo capítulo, aborda os conceitos, metodologias e técnicas de usabilidade e design de interfaces que irão sustentar o estudo e análise da aplicação Citydrive e, posteriormente, faz uma introdução às tecnologias e linguagens de programação estudadas que vão ser utilizadas e mencionadas ao longo do relatório, como por exemplo: XML, SOAP, REST, *Web Services* e Markdown.

O terceiro capítulo, descreve o primeiro projeto do estágio: a avaliação da usabilidade da aplicação Citydrive. Neste capítulo encontramos a análise feita a todos os ecrãs da aplicação e os erros apontados na mesma, tanto a nível de usabilidade como de funcionalidades e são apresentadas algumas soluções simples que solucionam parte dos problemas levantados anteriormente. Posteriormente, é feita uma análise à concorrência, onde mostra o estudo de quatro aplicações, comparando-as com a aplicação Citydrive e realçando os prós e os contras tanto das apps concorrentes como da app Citydrive.

O quarto capítulo, descreve o segundo projeto do estágio: a documentação da API. Neste capítulo é descrito todo o processo, desde a instalação e configuração dos programas necessários, às novas linguagens estudadas, aos novos tópicos e conteúdos criados e aos testes de código através dos *Web Services*.

PARTE I – CONTEXTUALIZAÇÃO

CAPÍTULO 1:

Caracterização da empresa

1.1 Mobiag

A Mobiag é uma empresa que acredita no futuro da mobilidade compartilhada e a missão deles é criar soluções inteligentes para a mobilidade urbana e potenciar os negócios da mobilidade com tecnologia de ponta (Mobiag – About us, 2017).

A Mobiag foi fundada com o objetivo de desenvolver ferramentas inovadoras para as empresas de mobilidade, seja *carsharing* ou aluguer, e ajudá-las a gerir as suas operações de forma mais eficiente e oferecer-lhes a oportunidade de moldar o futuro dos transportes (Mobiag – About us, 2017).

Esta startup, tem como alvo a inovação e implementação de novas formas de mobilidade que beneficiem a todos e criem soluções sustentáveis, por isso, oferece soluções flexíveis e sofisticadas o suficiente para alimentar qualquer negócio de mobilidade. Eles tencionam tornar as operações mais simples, fáceis e automáticas, tanto quanto possível, de maneira a que as empresas possam focar-se em oferecer o melhor serviço aos seus clientes (Mobiag – About us, 2017).

A Mobiag está localizada em Lisboa, Portugal e trabalha ativamente com dezenas de fornecedores da área da mobilidade que operam em vários continentes (Mobiag – About us, 2017).

1.2 Colaboradores

A equipa Mobiag é composta por 14 membros, alguns com várias décadas de experiência em várias empresas de mobilidade, consultoria e nas principais empresas de desenvolvimento de software, o que lhes dá uma certa vantagem para inovar nesta área (Mobiag – About us, 2017).

Na equipa de Gestão temos: João Félix (CEO), Rui Avelãs (*SVP Sales & Marketing*), Bernardo Mendonça (CTO e meu orientador de estágio) e João Pernes (*Head of Product Design and Customer Success*). Dos restantes 10 membros, só tive o prazer de trabalhar com os seguintes: João Filipe (Programador), Luís Azevedo (Logística), Marianna Vivo

(Contabilidade), Nuno Gomes (Gestor de Projeto) e Pedro Sousa (Design e Hardware) (Mobiag – About us, 2017).

1.3 Produtos

A Mobiag disponibiliza 4 produtos aos seus clientes:

- **App e webportal**: oferecem um conjunto de funcionalidades que são necessárias para o negócio da mobilidade e, caso o cliente já tenha uma solução, tem acesso total à API para integrar e construir por cima do que já possui (Mobiag – Mobiag Solution, 2017).
- **MobiCS:** baseado no software SAAS, este é um software desenvolvido para *carsharing* e negócios de aluguer. A partir daqui o cliente tem acesso a todas as operações (Mobiag – Mobiag Solution, 2017).
- **Mobiag Connect:** é uma caixa instalada em todos os veículos que constituem a frota de uma empresa. Este permite receber toda a informação sobre os veículos e controlá-los (Mobiag – Mobiag Solution, 2017).
- **Mobiag Network:** todos os clientes da Mobiag fazem parte duma grande rede. Isto significa que as frotas existentes, estão agregadas pelo mundo todo e são disponibilizadas a todos os clientes através do seu operador. Desta maneira, mesmo que um cliente viaje para um país onde o seu operador não exista, poderá continuar a ter acesso aos veículos e reservá-los sem ter que fazer download da aplicação local (Mobiag – Mobiag Solution, 2017).

1.4 Clientes

Os clientes da Mobiag distribuem-se por: Chile, México, Polónia e Portugal. No Chile, têm a Awto. No México, têm a Carrot. Na Polónia, têm a Leasing. E em Portugal, têm a Citydrive e a Hertz (Mobiag – Client, 2017).

1.5 Conceito *Carsharing*

O *carsharing* é um conceito que se tem vindo a tornar bastante conhecido e baseia-se num modelo de aluguer de carros, onde as pessoas podem alugar um

determinado carro por curtos períodos de tempo, que são taxados por minuto ou por hora. Este conceito tem atraído muitos utilizadores, uma vez que permite a alguém que não tenha veículo próprio, conduzir um carro ocasionalmente como também permite a quem já tenha um carro, conduzir um carro de um modelo diferente do que conduz no seu dia a dia (Wikipédia, 2014).

CAPÍTULO 2:

Enquadramento teórico

2.1 Definição de conceitos - Projeto 1

Para a realização de uma boa avaliação do design e da usabilidade de uma aplicação mobile e para conseguir compreender as metodologias que irão ser abordadas posteriormente, é necessário ter em mente algumas noções básicas sobre alguns conceitos como: usabilidade, usabilidade em dispositivos móveis, design de experiências (*UX Design*), técnicas de avaliação da usabilidade e design de interfaces (*UI Design*).

2.1.1 Conceito de usabilidade

Existem diferentes definições para o termo usabilidade, pois cada autor partilha de uma opinião diferente. No entanto, existem sempre alguns conceitos que sobressaem em relação a outros.

A *Internacional Standards Organization* (ISO 9241-11), defende que a usabilidade pode ser definida como a eficácia, eficiência e satisfação com que os utilizadores conseguem atingir objetivos específicos com um dado produto, num determinado contexto (tarefas, equipamentos, ambiente físico e social) (W3C, 2002).

- **Eficácia:** avalia se o utilizador alcançou os objetivos iniciais da interação, isto é, se conseguiu finalizar a tarefa com sucesso e se obteve bons resultados em termos de qualidade.
- **Eficiência:** refere-se à quantidade de esforço e recursos necessários para o utilizador alcançar um determinado objetivo. Os desvios e os erros que o utilizador realiza durante a interação, ajudam-nos a avaliar o nível de eficiência da interface.
- **Satisfação:** apesar de esta ser uma medida subjetiva, refere-se de uma maneira geral ao nível de conforto que o utilizador sente ao utilizar a interface e se este se sente satisfeito com os resultados obtidos dado os objetivos iniciais.

De acordo com Nielsen (2012), a usabilidade é definida como um atributo de qualidade relacionada com a facilidade com que os utilizadores aprendem a utilizar uma determinada interface. Além disto, Nielsen destaca 5 componentes de qualidade que, na opinião dele, definem a usabilidade:

- **Facilidade de aprendizagem:** a facilidade com que o utilizador interage com o sistema e realiza as tarefas no primeiro contacto com a interface;
- **Eficiência:** uma vez conhecendo o sistema, o utilizador deverá conseguir realizar as tarefas de forma rápida e eficiente;
- **Fácil memorização:** mesmo após algum tempo sem interagir com a interface, o utilizador deverá lembrar-se de como a mesma funciona;
- **Erros:** caso o utilizador cometa um erro, qual é a gravidade desse erro e com que facilidade consegue recuperar do mesmo;
- **Satisfação:** a interface deve transmitir confiança e fazer o utilizador sentir-se seguro;

Seja na web ou em mobile, a usabilidade é um conceito importante a ter em mente ao desenhar uma interface, pois se os utilizadores se deparam com um sistema com uma má usabilidade, simplesmente não vão querer interagir com o mesmo. Se uma aplicação for complicada de utilizar, se a informação que o utilizador pretende consultar não estiver facilmente acessível, se o utilizador não conseguir realizar as tarefas que pretende ou se simplesmente o utilizador se sentir perdido na aplicação, vai deixar de utilizá-la. Com certeza existirão outras aplicações que a substituam, pelo que o utilizador não se vai dar ao trabalho de estar a tentar aprender a interagir com um sistema de dificil compreensão e onde tem medo de estar sempre a cometer erros que não o deixam confortável e provocam stresse e insegurança numa situação que deveria ser simples e intuitiva.

Desta maneira, é importante que numa fase inicial do desenho de qualquer website/aplicação seja feita uma avaliação da usabilidade da mesma, não só porque o custo de modificar e melhorar o design é muito menos elevado numa fase inicial de desenvolvimento do que após o lançamento do website/aplicação no mercado, mas também porque iremos colocar no mercado uma aplicação útil e com boa usabilidade que qualquer pessoa conseguirá facilmente utilizar.

2.1.2 Usabilidade em dispositivos móveis

De acordo com Nielsen e Norman (2010), empresas como a Google e a Apple têm vindo a impor inovações a nível de interação gestual que vão contra os princípios da usabilidade em dispositivos móveis. A usabilidade em dispositivos móveis é definida pelo conceito dado anteriormente e ainda exige que os seguintes princípios sejam seguidos, o que não tem acontecido:

- ***Affordance*:** refere-se à perceção que temos de um determinado objeto que nos deixa a saber de imediato como é que o mesmo tem que ser utilizado. Quando a *affordance* de um objeto/aplicação corresponde realmente à sua real função, significa que o design da aplicação é bom e consegue ser usado mais eficientemente por parte dos utilizadores;
- **Feedback:** mostrar feedback ao utilizador sempre que este realiza uma ação na aplicação, isto é, enviar de volta algum tipo de mensagem ao utilizador para que este saiba que está a fazer tudo corretamente e pode prosseguir com a operação;
- **Consistência:** desenhar interfaces para terem operações semelhantes que usem elementos semelhantes para alcançar tarefas semelhantes. Por exemplo, se um determinado ícone representa determinada tarefa, esse mesmo ícone não pode ser utilizado para representar outra tarefa, pois vai resultar numa inconsistência e irá confundir o utilizador que irá ter 2 ícones iguais que realizam tarefas iguais;
- **Operações não-destrutivas:** não permitir que a aplicação realize ações que não possam ser canceladas e que não permitam ao utilizador corrigir algo que fez de mal e voltar atrás;
- **Descoberta:** a vantagem dos ecrãs não terem que ser memorizados pelo utilizador, uma vez que cada ação apresentada na interface podia ser descoberta através da exploração sistemática de menus;
- **Escalabilidade:** a aplicação deve estar preparada para funcionar em todos os tamanhos de ecrã existentes, sejam pequenos ou grandes;

- **Confiança:** o utilizador tem que ter a sensação que controla o sistema, pelo que cada ação que ele tente realizar através da interação gestual, deve funcionar sem exceção.

2.1.3 Design de Experiências *(UX Design)*

O design de experiências é o processo de melhorar a experiência do utilizador. Tem como objetivo tornar o produto útil, encontrável e acessível, que a sua interface seja usável e desejável, e que a informação que veicula seja credível (Morville, 2004). Este processo é normalmente divido em 3 fases:

1. **Design centrado no utilizador (UCD):** fase de investigação onde se analisa e antecipa o modo como os utilizadores vão usar o nosso produto (Allen, 2012). Nesta fase, são utilizados os seguintes métodos para estudar como os utilizadores percecionam, pensam e decidem (Weinschenk, 2011):
    - **Análise da concorrência (benchmarking):** estuda os produtos da concorrência e avalia a sua interface e usabilidade.
    - **Personas**: criação de personagens fictícias que ajudam os designers a perceber a necessidade dos utilizadores finais.
    - **Fluxos de utilizador**: diagramas que representam a rota do utilizador ao realizar determinadas tarefas (Allen & Chudley, 2012).
2. **Utilização da informação obtida na fase anterior** para a realização de:
    - **Wireframes:** diagramas de baixa fidelidade que representam o layout de um site ou aplicação. Permitem explorar e testar ideias de design numa fazer inicial dos projetos.
    - **Walkthroughs:** simulação das ações e pensamentos de um utilizador ao percorrer, passo a passo, os wireframes (Allen & Chudley, 2012).
3. **Recolha de feedback e testes de usabilidade:** após a fase do design, chega a fase de recolha de feedback por parte da equipa do produto e do cliente e passamos a recorrer também aos testes de usabilidade com utilizadores, que nos ajudam a emendar os problemas de usabilidade da aplicação.

2.1.4 Técnicas de avaliação da usabilidade

Existem vários métodos para estudar e avaliar a usabilidade, mas o mais comum e mais utilizado é o teste com utilizadores (Nielsen, 2012).

Este teste é realizado reunindo um grupo de utilizadores (5 utilizadores é normalmente o suficiente) que possuam ou não conhecimento sobre a aplicação. Este utilizadores irão, individualmente, realizar um conjunto de tarefas pré-definidas pela equipa de desenvolvimento da aplicação que, por norma, correspondem às funcionalidades principais que queremos que a aplicação tenha. Durante esta etapa, o nosso papel será observar a reação dos utilizadores durante a interação com a interface e pedir que estes apontem as dificuldades que estão a ter sem nunca interferirmos no processo, isto é, caso o utilizador tenha alguma dúvida sobre como realizar determinada tarefa e nos questione, não podemos responder, pois estaremos a influenciar e a contaminar o resultado do teste, uma vez que nós já sabemos a resposta e estamos a testar se o que fizemos é percetível. Se para os utilizadores, realizar essa determinada tarefa não é algo óbvio e intuitivo, então temos um problema de usabilidade. Através das dificuldades e problemas apontados pelos utilizadores, arranjamos soluções para os mesmos e voltamos a realizar testes com novos utilizadores para filtrar e encontrar novos os problemas de usabilidade.

Como resultado, iremos desenvolver uma aplicação com um ótimo nível de usabilidade que, pelas definições anteriores, podemos concluir que é a facilidade e rapidez que o utilizador tem em interagir com a interface e lembrar-se de como a mesma funciona sem cometer erros e sem sentir-se inseguro.

No entanto, podemos distinguir mais técnicas de avaliação da usabilidade, segundo Cybis (2003), tais como:

- **Técnicas prospetivas:** procuram a opinião do utilizador sobre a interação com o sistema através, por exemplo, da elaboração de questionários;
- **Técnicas preditivas:** procuram prever os erros presentes nas interfaces sem a intervenção direta de utilizadores como, por exemplo, através da avaliação heurística;
- **Técnicas objetivas:** procuram constatar os problemas a partir da observação do utilizador a interagir com o sistema através de ensaios de

interação ou monitorização dos utilizadores durante os testes através de algum software.

Neste projeto, foi utilizada a técnica preditiva. Isto é, foi feita uma avaliação heurística à aplicação Citydrive. A avaliação heurística foi desenvolvida por Nielsen (1990) e, como referido anteriormente, esta técnica de avaliação não inclui a participação de utilizadores. Por norma, é realizada com um grupo de 3 a 5 utilizadores especialistas em ergonomia, com experiência e competência no assunto que, individualmente, procuram identificar os problemas da interface. No fundo, é um julgamento de valor sobre as qualidades ergonómicas das interfaces humano-computador (Cybis, 2003).

Existem 10 regras de heurística elaboradas por Nielsen (1995) que servem de referência na avaliação de websites e aplicações, que são apresentadas a seguir:

- **Visibilidade do sistema:** O sistema deve informar sempre o utilizador sobre o que ele está a fazer no momento de maneira a que o utilizador tenha sempre algum feedback do sistema e saiba se está a agir corretamente ou o que se está a passar com a aplicação.
- **Falar a linguagem do utilizador:** a terminologia da interface deve ser baseada na linguagem do utilizador e não orientada ao sistema. As informações devem ser organizadas conforme o modelo mental do utilizador.
- **Liberdade e controlo:** o utilizador deve sentir que pode controlar a interface, devendo ser possível sair facilmente das mais variadas situações como, por exemplo, ser capaz de cancelar uma tarefa a qualquer momento, desfazer uma determinada operação, voltar ao passo anterior e, a partir de qualquer página do website/aplicação, ser possível voltar à *homepage*. Isto deixará o utilizador confortável e à vontade para cometer erros e lidar com o sistema, uma vez que irá aprender que pode desfazer os eventuais erros.
- **Consistência e standards:** este é um dos princípios básicos da usabilidade. A mesma operação deverá ser apresentada sempre na mesma localização em todos os ecrãs e deverá ser formatada da mesma maneira para facilitar o reconhecimento.

- **Prevenção de erros:** conhecer as situações que mais provocam erros e desenhar cuidadosamente a interface de maneira a prevenir a ocorrência de erros.
- **Minimizar a sobrecarga de memória do utilizador (*recall vs. recognition*):** a interface deve exibir elementos de diálogo e permitir que o utilizador faça as suas escolhas sem obrigá-lo a lembrar-se de como os comandos funcionam ou o que fazem. As pessoas são melhores a reconhecer do que a se recordar de tarefas que executaram previamente. A memória funciona melhor através do reconhecimento porque o reconhecimento funciona através da experiência enquanto a recordação funciona através da aprendizagem.
- **Flexibilidade e eficiência de utilização (atalhos):** a interface deve disponibilizar atalhos para utilizadores mais experientes para que este possam realizar as operações mais rapidamente. Por exemplo, teclas de função, duplo clique no rato e botões programáveis para funções frequentes.
- **Estética e design minimalistas:** a interface deve ser o mais simples possível. Deve ser apresentada somente a informação que o utilizador precisa, uma vez que o excesso de informação irá resultar na diminuição da visibilidade da informação relevante.
- **Boas mensagens de erro:** as mensagens de erro devem ter uma linguagem clara, precisa e sem códigos de maneira a ajudar o utilizador a resolver facilmente o problema e sem intimidá-lo ou culpá-lo pelo erro.
- **Ajuda e documentação:** apesar de ser melhor o utilizador conseguir usar o sistema sem ajuda ou documentação, se necessário, esta ajuda deve estar facilmente acessível e deve ser direta e pouco extensa.

2.1.5 Design de Interfaces *(UI Design)*

O design de interfaces foca-se em garantir que os elementos da interface são fáceis de aceder, compreender e usar, no sentido de facilitar as tarefas que os utilizadores necessitam de realizar (AA. VV., n.a). O design de experiências é parte integrante do design de interfaces e vice-versa. Alguns métodos de *UX Design*, como os *wireframes*, podem também ser utilizados em etapas do *UI Design*. No entanto, o *UI Design* é sempre mais associado a uma fase onde o aspeto da interface começa a ganhar mais fidelidade, o que não significa que o UI Design seja baseado em protótipos de alta-fidelidade. A designer Helga Moreno, cita o seguinte: *"Something that looks great but is difficult to use is exemplary of great UI and poor UX. While something very usable that looks terrible is exemplary of great UX and poor UI."* (Emil, 2017).

2.2 Definição de conceitos – Projeto 2

Para ser capaz de testar o código presente na documentação da API da Mobiag tive primeiro que estudar alguns conceitos e linguagens utilizados nesta documentação, que passo a explicar posteriormente, tais como: API, HTTP, XML, SOAP e REST. E para ser capaz de criar, editar, corrigir e eliminar conteúdo tive que estudar a sintaxe da linguagem Markdown, pois era a linguagem utilizada para a documentação.

2.2.1 O que é uma API

API é um acrónimo de *Application Programming Interface* que, em português, se traduz em Interface de Programação de Aplicações. Esta interface é definida como um conjunto de definições, protocolos e ferramentas utilizadas para construir uma aplicação. É um conjunto de métodos de comunicação definidos entre vários componentes de software. Enquanto utilizamos um software, a interface funciona por trás de tudo sem que os utilizadores se apercebam, estabelecendo conexões entre outros sistema ou aplicações (Wikipédia).

Uma boa API torna mais fácil desenvolver um programa ou aplicação, pois esta já fornece todos blocos de código necessários ao programador. Uma API pode ser utilizada em diversas aplicações e serve para vários fins como sistemas web, sistemas operativos, base de dados, hardware de computador ou bibliotecas de software (Wikipédia).

Um exemplo de uma API comum e bastante utilizada é a do Facebook. Hoje em dia, muitas das aplicações que instalamos nos nossos telemóveis ou os websites que acedemos online, permitem-nos fazer o registo através do Facebook. Estas aplicações e websites, utilizam a API do Facebook para tornar possível o login através do mesmo e, nesta situação, os programadores simplesmente têm que consultar a API do Facebook e documentação da mesma e efetuar a ligação com a aplicação ou website de acordo com as regras da API (Facebook Developers).

As APIs, quando utilizadas num contexto de desenvolvimento web são definidas por um conjunto de pedidos em HTTP e respostas, normalmente, em XML ou JSON. Estas são as Web APIs que são virtualmente conhecidas por *Web Services*. Os *Web Services* comunicavam, por norma, através do SOAP, mas com a chegada da Web 2.0, o SOAP tem deixado de ser utilizado e tem sido substituído por REST (Wikipédia).

2.2.2 O que é HTTP

HTTP (*Hypertext Tranfer Protocol)* é um protocolo para sistemas de informação que funciona através de um conjunto de pedidos-respostas entre clientes e servidores e é a base da comunicação para a World Wide Web. Estes pedidos-respostas são feitos através de métodos pré-definidos do HTTP como o GET e o POST (IETF Tools e W3Schools).

O GET faz um pedido de informação a uma determinada fonte enquanto o POST submete a informação a para ser processada para uma determinada fonte (W3Schools).

Um exemplo de como funciona o HTTP: um cliente (browser) submete um pedido HTTP para o servidor. O servidor retorna a resposta ao cliente. A resposta contém o estado da informação do pedido e poderá também conter o conteúdo pedido (W3Schools).

2.2.3 O que é o XML

XML *(eXtensible Markup Language)* foi desenhada para ser legível tanto por humanos como por máquinas, o que faz com que esta linguagem seja uma vantagem, pois é uma linguagem independente de software e hardware, o que significa que não há incompatibilidade na troca de informação entre diferentes sistemas e pode ser facilmente expandida ou atualizada para novos sistemas operativos, aplicações e browser sem que haja perda de informação. É utilizada para guardar e transportar informação (W3Schools).

O XML, só por si, não faz nada. É somente informação dentro de tags que precisa de ser processada por um software que permita enviar, receber, guardar ou mostrar essa informação (W3Schools).

Ao contrário do HTML *(Hypertext Markup Language)*, o XML não tem tags pré-definidas. As tags e a estrutura do XML são inventadas na hora pelo autor (W3Schools).

Os documentos XML têm uma estrutura de elementos em árvore. Uma árvore XML começa sempre com o elemento ***root*** que é o ***parent*** de todos os outros elementos. A partir daqui ramificam-se os elementos ***child***, que são os elementos que

se encontram exatamente um nível abaixo do elemento ***root***. Podemos ainda ter sub-elementos (W3Schools) **(Fig.1)**.

2.2.4 *Web Services*, SOAP e REST

Pelo que foi mencionado anteriormente, podemos concluir que um *Web Service* é qualquer serviço que esteja disponível e acessível através da Internet ou redes privadas, que use a linguagem XML para o sistema de pedidos-respostas, que não esteja vinculado a nenhum sistema operativo ou linguagem de programação e que se auto-descreva através da gramática XML (Tutorials Point).

A plataforma de *Web Services* é uma API baseada em XML e HTTP e todos os *Web Services* padrão utilizam as seguintes componentes (Tutorials Point):

- **SOAP - *Simple Object Access Protocol*:**
  - Protocolo de comunicação de aplicações.
  - É um formato utilizado para o envio e receção de mensagens.
  - É baseado em XML.
  - É uma plataforma independente.
  - Uma mensagem SOAP contém sempre estes elementos na sua estrutura: <envelope>, <header>, <body>, <fault>.(FIGURA)
- **WSDL – *Web Services Description Language:***
  - É escrito em XML**.**
  - É utilizado para descrever os *Web Services*, especificando a localização e métodos do serviço através destes elementos: <types>, <message>, <portType> e <binding>. (FIGURA)

Os *Web Services* funcionam, assim, com a ajuda do XML que identifica a informação, com o SOAP que transfere a informação e com WSDL que descreve a disponibilidade do serviço pedido (Tutorials Point).

Além destes *Web Services*, existe ainda outro tipo de *Web Service* que é baseado na arquitetura REST *(Representational State Transfer)* e denomina-se RESTful Web Service. Os serviços RESTful apresentam várias vantagens por serem leves,

altamente escaláveis e de fácil manutenção e, como os anteriores, são muito usados para criar APIs para aplicações baseadas na Web (Tutorials Point).

Em REST, as comunicações continuam a ser estabelecidas através dos métodos HTTP, mas distingue-se dos *Web Services* anteriores no sentido em que tudo aqui gira em torno de recursos (Tutorials Point).

Cada componente é um recurso e esse recurso é acedido através duma interface que usa os métodos HTTP. O servidor REST fornece acesso aos recursos e o cliente REST acede e mostra os recursos. Cada recurso é identificado por um URI *(Uniform Resource Identifier)* (Tutorials Point).

REST pode utilizar vários formatos para representar os seus recursos como: Text, HTML, XML e JSON. Este último é, atualmente, o formato mais utilizado nos *Web Services* (Tutorials Point).

2.2.5 Sintaxe Markdown

Markdown é uma ferramente de conversão de texto para HTML para escritores web. O Markdown é definido por duas coisas: pela maneira fácil e simples de ler e escrever no formato *plaintext* e pela ferramenta de software, que é escrita em Perl, e converte o *plaintext* para uma estrutura válida de XHTML/HTML (Daring Fireball).

O objetivo principal do design da sintaxe Markdown é ser o mais legível possível, isto é, um documento Markdown deve poder ser publicado "*as-is"*, o que significa que não deverá apresentar sinais de formatação ou tags, como no HTML (Daring Fireball).

Desta maneira, a sintaxe do Markdown é composta somente por carateres de pontuação que foram escolhidos de modo a irem de encontro à sua função e significado. Por exemplo, se quisermos por uma palavra a negrito, acrescentamos um asterisco no início e no fim da palavra: input - *negrito*, output – **negrito**. Se quisermos um título que em HTML é representado pelas tags <h1></h1>, no Markdown simplesmente escrevemos: #Título (Daring Fireball).

PARTE II – PROJETOS

CAPÍTULO 3:

Citydrive – Avaliação da usabilidade

3.1 O que é a Citydrive e como funciona?

A Citydrive é a maior marca de *carsharing* em Portugal que disponibiliza carros ao trajeto e ao minuto. Existem desde 2014 e introduziram uma grande novidade no mercado, o modelo *free-floating*, o que significa que pela primeira vez em Portugal o utilizador já não precisa de devolver o carro ao local de origem (Citydrive – Sobre nós, 2015).

O conceito e a aplicação da Citydrive resumem-se em 5 passos (Citydrive – Como funciona, 2015):

- **Passo 1 – Aderir à Citydrive:** Fazer o registo através do Website deles e, caso tenha smartphone, instalar a aplicação Citydrive no mesmo;
- **Passo 2 – Localizar um carro:** Através de um dispositivo móvel ou de um PC, pesquisar e encontrar os carros mais perto de nós;
- **Passo 3 – Reservar um carro:** Dos carros disponíveis, escolher o que preferimos e reservá-lo premindo o botão "Reservar" e inserindo o PIN definido no registo e o carro fica imediatamente disponível para utilização;
- **Passo 4 – Começar a conduzir:** Ao chegar ao carro, abrir as portas através da app no smartphone ou através do cartão de acesso atribuído a quem não possuí smartphone. Dentro do carro, verificar o carro e o diagrama e reportar danos novos caso existam. Inserir novamente o PIN para activar o carro. Retirar a chave do conector e começar a conduzir;
- **Passo 5 – Estacionar:** Para terminar a viagem, basta estacionar numa das zona de serviço permitidas, colocar a chave no conector e sair do carro. Uma vez fora do carro, trancar as portas através da app no smartphone ou através do cartão. Caso queiramos reservar o carro, trancamos as portas com a chave e levamos connosco.

O custo da viagem é posteriormente descontado, do cartão de crédito previamente adicionado ou do saldo carregado na conta de cliente.

3.2 Estudo da aplicação Citydrive

Durante os meus primeiros dias de estágio, comecei por estudar o conceito da empresa Citydrive e compreender o que era suposto a aplicação deles fazer ou não e, desta maneira, consegui começar a testar as funcionalidades da mesma e avaliá-la a nível de usabilidade.

Durante os testes encontrei vários erros, uns de usabilidade que passo a descrever a seguir, enquadrando-os dentro das regras heurísticas às quais estes não iam de encontro e outros erros técnicos e funcionais que explico posteriormente.

Para o leitor poder compreender melhor o funcionamento da aplicação, passo a descrever o *workflow* da aplicação: após realizarmos o registo no Website da Citydrive (que não é disponibilizado na aplicação), instalamos a aplicação **(Fig.2)**. O primeiro ecrã é o de arranque, que mostra o logótipo da Citydrive e, em seguida, o formulário de login **(Fig.3)**. Após efetuarmos o login, somos apresentados com um tutorial, que ocupa todo o ecrã e explica as funcionalidades da aplicação **(Fig.4)**. Este tutorial só é mostrado da primeira vez que ligamos a aplicação e, se quisermos consultá-lo novamente, está disponível no Menu **(Fig.5)**. Podemos decidir fechar o tutorial a qualquer altura, sem necessidade de vê-lo até ao fim.

Depois do tutorial, somos apresentados com o ecrã inicial da aplicação que é um mapa que nos indica automaticamente qual é o carro que está mais próximo de nós e desenha o trajeto desde a nossa localização até lá. Neste mesmo ecrã, é apresentada informação sobre o carro, um ícone que mostra o estado da reserva, um botão de menu, um botão de pesquisa e um botão para efetuar a reserva **(Fig.6)**.

Ao clicarmos no botão de Menu, abre-se um Menu lateral da direita para a esquerda, onde podemos aceder às seguintes secções: Início, Viagens, Faturação, Promoções, Códigos Promocionais, Conta, Tutorial e Contactos **(Fig.5)**.

Ao clicarmos no botão de Reserva no ecrã inicial, é-nos pedido o PIN que definimos durante o registo **(Fig.7)**. De seguida, se a informação inserida estiver correta e já tivermos chegado ao carro, somos reencaminhados para um ecrã que inicia o processo de Reserva, onde podemos então desbloquear as portas do carro,

verificar se existem novos danos no carro e registá-los na aplicação e, então, teremos acesso à chave e podemos começar a conduzir.

No botão de pesquisa, abre-se uma lista de todos os carros disponíveis na nossa zona e mostra informação do mesmo, como o nível de combustível, matrícula, etc **(Fig.8)**. Se quisermos algum carro em específico, podemos pesquisar através da matrícula e selecionar da lista devolvida o carro que desejamos que irá aparecer, posteriormente, no mapa.

Erros de usabilidade encontrados:

1. No tutorial que aparece quando abrimos a aplicação, aparecem umas setas que não são clicáveis mas que sugerem ao utilizador que é para mudar de ecrã clicando lá. Isto não acontece, uma vez que para mudar o ecrã temos que fazer *swipe* para a esquerda ou direita **(Fig.9)**.
2. Depois de clicarmos nos Termos de Utilização do mapa, não é possível voltarmos atrás **(Fig.10)**.
3. A imagem do botão "Reservar" que aparece no tutorial não é consistente com o que aparece no ecrã principal da aplicação, uma vez que o do tutorial está a azul e o da aplicação a verde **(Fig.11)**.
4. É impossível perceber o que significa o ícone "sem viagens activas" se o utilizador não vir o tutorial **(Fig.12)**.
5. Não existe um botão *Home* presente em todos os ecrãs e de fácil acesso. São sempre necessários dois cliques: Menu > Início **(Fig.13)**.
6. Em Menu > Conta, é transmitida a ideia de ser possível fazer upload dos nossos dados, mas isso não acontece (**Fig.14)**.
7. Os ícones do Menu e Pesquisa de carros são demasiado parecidos **(Fig.15)**.
8. O ícone de Pesquisa de carros é pouco explícito e muitas vezes a visibilidade é pouca como tem o mapa como fundo **(Fig.16)**.
9. O ícone de detalhes da viagem é igual ao ícone que representa o motor em funcionamento **(Fig.17)**.
10. O ícone de carro reservado não é percetível **(Fig.18)**.

11. Após tentarmos realizar uma reserva sem termos Saldo na nossa conta de cliente, aparece simplesmente uma mensagem que diz “A sua conta foi suspensa” **(Fig.19)**.
12. Após clicarmos numa das opções disponibilizadas no Menu, o Menu desaparece e somos reencaminhados para onde clicamos, mas não aparece nenhum título ou algo que nos indique/lembre em que parte do Menu nos encontramos **(Fig.20)**.
13. Se não tivermos o GPS activo ao ligar a aplicação, aparece uma mensagem a dizer “GPS indisponível” e não conseguimos mexer na aplicação nem fechar a mensagem. É preciso fechar e abrir a aplicação, ligando antes o GPS **(Fig.21)**.
14. São necessários muitos passos para realizar o Logout: Menu > Contas > Pessoal > Logout **(Fig.22)**.

Erros que desrespeitam a regra de heurística:

- Visibilidade do sistema: Erro 12
- Falar a linguagem do utilizador: Erros 4, 7, 8, 10, 11 e 13
- Liberdade e controlo: Erros 2 e 13
- Consistência e standards: Erros 1, 3, 7, 8, 9 e 10
- Prevenção de erros: Erros 1, 11 e 13
- Minimizar a sobrecarga de memória do utilizador: Erros 4 e 14
- Flexibilidade e eficiência de utilização: Erro 5
- Estética e design minimalistas: Erro 6
- Boas mensagens de erro: Erros 11 e 13

Durante esta análise deparei-me com outros tipos de erros que não se relacionavam tanto com a usabilidade mas sim com as funcionalidades da aplicação, que muitas vezes estavam mal implementadas ou faltavam implementar.

Alguns desses erros foram:

- Caso o GPS não estivesse a funcionar corretamente ou não estivesse a ser detetado apesar de estar ligado, não existia maneira da aplicação saber onde nos encontrávamos e mostrar o carro mais próximo.
- No Menu > Conta > PIN, o PIN estava visível e, por questões de segurança, não devia.
- Não era possível registar-se como novo utilizador através da aplicação. Só através do website.
- Não era possível alterar os dados pessoais na aplicação, apesar dos dados serem apresentados numa parte da aplicação: Menu > Conta > Pessoal.
- As zonas de serviço (Verde, Amarelo, Azul) também deveriam ser representadas no mapa da aplicação, como no website, para os utilizadores saberem onde podem estacionar.
- Ao selecionar um carro deveria aparecer o preço taxado por minuto ou então ao clicar Reservar, como acontece no website.
- Deveria ser possível filtrar/pesquisar pela marca do carro, preço por minuto, distância a que se encontra e classe (citadino, desportivo, etc).
- No Menu > Viagens, existe a opção de viagens agendadas, mas a aplicação não permite agendar viagens (ainda) **(Fig.23)**.
- Não existe a opção de efetuar o carregamento de saldo através da aplicação. É necessário utilizar o website.

Soluções sugeridas para a correção destes erros (usabilidade e funcionalidades):

- No tutorial, fazer as setas clicavéis e remover o *swipe* ou manter o *swipe* mas retirar as setas do design do tutorial e substituir por "pontos". Cada ponto representava um ecrã e fazíamos *swipe* e a cor do ponto do ecrã onde nos encontrávamos mudava de cor, como é comum nas aplicações atuais.
- Caso o GPS não conseguisse detetar a nossa localização, ter um campo para introduzir o nosso endereço ou ser possível marcar no mapa onde nós estávamos colocando um pin.

- Ao abrir os Termos de Utilização, reencaminhar para o browser e, caso seja necessário abrir mesmo na aplicação, colocar um botão de voltar atrás ou os Termos num pop-up que possa ser fechado.
- Pôr todos os botões apresentados no tutorial a verde para ficarem consistentes com o design da aplicação.
- Substituir o campo do PIN por ****.
- O preço por minuto poderia aparecer no dropdown que já existe no ecrã inicial da aplicação, onde são apresentadas algumas informações sobre o carro.
- Substituir todos os ícones poucos explícitos ou iguais mas com funções diferentes, por ícones que representem visualmente a função a ser desempenhada por aquele botão.
- Acrescentar um título a cada ecrã que existe no menu, de maneira a que o utilizador saiba sempre onde se encontra. Por exemplo, se clicarmos em Menu > Viagens, no ecrã Viagens aparece um título "Viagens" e assim o utilizador já sabe, caso se esqueça onde clicou, onde está.
- Dentro de cada opção do Menu, existem separadores que permitem alternar entre vários ecrãs com diferentes tipos de informação dentro de um mesmo ecrã geral. Para alternar entre estes separadores achei que seria útil implementar a função de fazer *swipe*, uma vez que facilitava a interação e substituia o facto do utilizador ter que clicar em cada separador para mudá-lo.
- Acrescentar um botão *Home* em todos os ecrãs existentes ou utilizar o logótipo, que aparece em todos os ecrãs, para servir de botão *Home*, como acontece em muitos websites.
- Ao tentar efetuar uma reserva sem ter saldo, mostrar uma mensagem mais subtil e menos intimidante ao utilizador como, por exemplo: "De momento não é possível reservar a sua viagem, pois encontra-se sem saldo na sua conta de cliente. Carregue o seu saldo e tente novamente.".
- Acrescentar o botão de Logout no fim do Menu e retirá-lo de Conta.
- Ao entrar em Conta, pedir o PIN ao utilizador.

Apesar de muitos destes erros parecerem facéis de corrigir, na altura que realizei a análise da aplicação, esta já se encontrava no mercado e muitas funções já estavam implementadas de certa forma o que dificulta, posteriormente, fazer alterações no código e no design da aplicação. O facto de não ter sido feito um estudo da usabilidade por parte de um designer na fase inicial do projeto, condicionou bastante a usabilidade e as funcionalidades da aplicação que agora fica limitada em certas situações pelo facto de se tornar demasiado moroso e dispendioso estar a refazer o código.

3.3 Análise da concorrência: *Car2Go* & *DriveNow*

De maneira a ter um termo de comparação no que diz respeito ao *workflow* e às funcionalidades da aplicação Citydrive, estudei duas aplicações concorrentes que já existiam no mercado, também no ramo de *carsharing*: *Car2Go* e *DriveNow.*

Observei, então, as vantagens, diferenças e funcionalidades que estas aplicações tinham em relação à Citydrive.

Observações da aplicação *Car2Go*:

- **Tutorial:** Ao abrir a app pela primeira vez aparece logo o tutorial. No fim do tutorial aparece então o login **(Fig.24)**.
- **Login:** A interface de login é pop-up que aparece por cima do mapa, que é o ecrã principal **(Fig.25)**.
- **Registo:** É constituído por 3 passos obrigatórios – Dados pessoais, Documentação (Carta de condução, etc), Dados de pagamento.
- **Funcionamento geral da aplicação:**
  - Após o login aparece o mapa com o botão de menu, barra de pesquisa, botão de localização, botão com lista de carros e botão de radar **(Fig.26)**.
  - No mapa aparecem todos os carros disponíveis na zona **(Fig.26)**.
  - Ao clicar num dos carros, aparece a matrícula, morada do carro, opção de reservar, opção de alugar na hora, preço por minuto e marca do carro **(Fig.26)**.

- Ao expandir esta informação, aparece ainda o nível de combustível, a distância em km a que nos encontramos do carro e a foto do carro **(Fig.26)**.
- Existe um campo para inserir o endereço em que nos encontramos **(Fig.26)**.
- Indica qual é o carro mais próximo de nós e quanto tempo demoramos até chegar lá.
- Existe uma lista dos carros disponíveis e dos carros disponíveis em radar caso definidos **(Fig.27)**.
- É possível definir um radar **(Fig.28)**.
- No Menu existe uma seção Help *> FAQ, Legal Information, Privacy Policy* **(Fig.29)**.
- É possível ativar nas definições a opção de aparecer as bombas de gasolina no mapa **(Fig.30)**.
- É possível mostrar ou ocultar as zonas de serviço **(Fig.30)**.
- É possível pôr o mapa em modo street ou satélite **(Fig.30)**.
- É possível pôr as unidades em KM ou Miles **(Fig.30)**.
- Mostra os parques de estacionamento no mapa e ao clicar num deles aparece quantos estacionamentos estão disponíveis para *carsharing* e quantos deles estão livres.
- Possibilidade de sair do mapa da app e ir para o Google Maps.
- Fazendo zoom out sinaliza todas as cidades que têm o serviço de *carsharing* disponível **(Fig.31)**.
- É possível filtrar os carros que queremos que apareçam ou não, no mapa.

Observações da aplicação *DriveNow*:

- **Tutorial:** Tutorial aparece ao abrir a aplicação pela primeira vez. O botão de login está sempre presente na parte inferior do ecrã. Parte superior constituída por um slide de imagens com pequenas explicações sobre a app **(Fig.32)**.

- **Registo:** O registo é feito no website da *DriveNow* e não na aplicação. Tem 3 passos: Conta, Contactos, Dados de Pagamento.
- **Funcionamento geral da aplicação:**
  - Após o login aparece o mapa com os carros disponíveis. Neste ecrã existem 3 botões: um para filtrar, outro para ativar/desativar a zona de serviço e o botão de localização **(Fig.33)**.
  - Mostra o carro que se encontra mais perto de nós, o trajeto e o tempo que demoramos até o mesmo.
  - Após selecionar um carro, aparece a distância, nível de combustível/bateria e se é automático ou manual e dá-nos a opção de reservar. Podemos escolher ver mais detalhes, onde podemos ver a matrícula, localização, preços, condições do carro, o tipo de combustível e onde podemos ainda reportar algum problema **(Fig.34)**.
  - No Menu existe uma seção *How it works, FAQ, Terms and Conditions* e *Privacy Policy* **(Fig.35)**.
  - Nos filtros é possível filtrar por marca de carro, por carros de combustão ou elétricos e por nível de combustível/bateria **(Fig.36)**.
  - Disponibiliza uma lista dos carros disponíveis **(Fig.37)**.
  - É possível definir radar **(Fig.38)**.
  - Existe um campo para inserir o endereço em que nos encontramos **(Fig.39)**.
  - É possível mudar a cidade em que nos encontramos através da lista de cidades disponíveis **(Fig.40)**.
  - Quando o mapa está minimizado aparece o número de carros naquela zona/rua no símbolo do carro **(Fig.41)**.
  - É possível mostrar/ocultar os favoritos, bombas de gasolina e estações de carregamento de carros **(Fig.42)**.

Após a análise destas duas aplicações, consegui concluir que existem vários pormenores e funcionalidades que a aplicação Citydrive deveria ter, que fariam toda a diferença na interação dos utilizadores com a mesma, uma vez que certos aspetos irião

contribuir para o aumento da usabilidade e acessibilidade, já que alguns aspetos fazem com que a aplicação possa ser utilizada por uma maior variedade de pessoas.

Funcionalidades sugeridas a serem introduzidas na aplicação Citydrive:

- Adicionar a funcionalidade Radar, que permite ao utilizador definir um horário e um raio de, por exemplo 5km, e é notificado sempre que aparecer um carro dentro dessa distância a partir da hora definida.
- Ser possível filtrar os carros por marca, preço e distância.
- Ser possível mostrar/ocultar bombas de gasolina, zonas de serviço e parques de estacionamento.
- Mostrar uma lista de carros e cidades/países onde é possível fazer a reserva de carros.
- Disponibilizar uma opção nas definições que nos deixa alterar as unidades de KM para Miles.
- No Menu, acrescentar uma secção Ajuda > FAQ e Tutorial.
- No Menu, acrescentar uma secção de *Legal Information* > *Privacy Policy, Terms and Conditions* e Contactos.
- Tornar possível o registo através da aplicação em vez de ser somente através do website.
- Uma vez que o ponto anterior seja possível, permitir que a informação pessoal possa ser editada e atualizada também através da aplicação.
- Adicionar a funcionalidade Agendar Viagem, ou seja, definir uma frota de carros que faça somente viagens agendadas e permitir ao utilizador escolher um desses carros e agendar a utilização do mesmo para um dia e hora à sua escolha.
- Durante o registo, permitir saltar o passo de adicionar a informação de pagamento e deixar o utilizador inserir essa informação, posteriormente, na área de cliente.
- Mostrar um tutorial mais atrativo e dinâmico, que não seja à base de imagens estáticas. Fazer, por exemplo, uma animação para cada ecrã e cada ecrã explica passo a passo como é que o utilizador deve interagir com a aplicação.

- Caso o tutorial se torne muito extenso e o utilizador queira consultá-lo posteriormente, apresentá-lo no Menu > Ajuda > Tutoriais, mas dividi-los por tópicos de maneira a que o utilizador possa aceder diretamente à parte do tutorial na qual tem dúvidas.

CAPÍTULO 4:

Documentação da API

4.1 Instalação e acesso remoto

Para conseguir aceder à documentação da API da Mobiag, de maneira a poder criar e editar conteúdo, tive que configurar o editor de texto, que no meu caso foi o Sublime Text 3, para aceder remotamente ao servidor onde a documentação estava alojada.

Após o *download* e instalação do Sublime Text 3, passei a consultar a documentação do mesmo para saber como configurar uma ligação SFTP *(Secure File Transfer Protocol).* Fiquei a saber que tinha que fazer o *download* do *Control Package*, colocar o ficheiro no diretório de *packages* do Sublime **(Fig.43)** e instalá-lo no Sublime através da palete de comandos, escrevendo e selecionando a opção *Package Control: Install Package* **(Fig.44)**. Posto isto, aparece uma lista de comandos, dos quais tive que escolher instalar o SFTP Sublime (Sublime – *Installation*) **(Fig.45)**.

Uma vez instalado o *package*, tive que ir a File > SFTP/FTP > Setup Server **(Fig.46)**. Esta função cria automaticamente um ficheiro onde temos que colocar os dados de acesso ao servidor, ou seja, o tipo de ligação (SFTP), nome do *host*, nome do *user*, *password* e o caminho remoto onde se encontra o nosso projeto no servidor **(Fig.47 e 48)**. Ao salvar, damos o nome que quisermos ao ficheiro, que neste caso foi *Slate* e este é guardado num diretório do Sublime (Sublime – *Usage*) **(Fig.49)**.

Agora, para termos acesso a todos os ficheiros da documentação, basta irmos à palete de comandos e escrevermos *Browse Server* e escolher o nome do servidor que acabamos de configurar e teremos acesso imediato a todos os ficheiros necessários **(Fig.50 e 51)**. Ao seleccionar um ficheiro, basta escolher a ação que desejamos efetuar como editar, mover ou eliminar **(Fig.52 e 53)**.

Feita a configuração, podemos editar e criar conteúdo à vontade e salvar, que o upload da informação é feito automaticamente para o servidor **(Fig.54)**.

4.2 Introdução à API Mobiag

Esta documentação é utilizada frequentemente pelos clientes da Mobiag que compram os seus serviços, daí a exigência de uma boa documentação, pois pode vir a ser utilizada por vários tipos de utilizador, com pouca ou muita experiência nesta área, que têm que ser capazes de compreender minimante o que se passa na API para poder testá-la e aplicá-la aos seus objetivos.

Com esta API é possível testar vários serviços, como: pesquisar os carros disponíveis, reservar um carro, desbloquear portas, desligar o imobilizador, relatar os danos, consultar valores de viagem, ligar o imobilizador, trancar as portas e terminar a viagem. A documentação ajuda o utilizador a perceber os serviços que são utilizados para comunicar com os carros e com o *backoffice*.

A documentação da API da Mobiag encontra-se online e é acessível através deste link: http://api.mobiag.com:4567. O design da documentação é baseado num *template* denominado Slate que está no repositório Github. No Slate, o conteúdo/código é escrito na sintaxe *Markdown* e a sua estrutura está dividida em três colunas que, na documentação da Mobiag, representam o seguinte (Github – *Documentation*) **(Fig.55)**:

1. **Coluna esquerda**: onde se encontra o logótipo, caixa de pesquisa e o menu, que engloba as seguintes secções: *Introduction, Get Started, Authentication, Car Club Data, Customer, Tags, Password Recovery, Search Cars, Booking a Car, Report Damage, Operator* e *Subscriptions*.
2. **Coluna central**: onde é mostrado o conteúdo da secção escolhida no menu.
3. **Coluna direita**: é onde se encontram os códigos necessários para testar os serviços e são mostrados também consoante a secção escolhida no menu. Esta coluna divide-se em dois separadores: o SOAP e o REST.

4.3 Familiarização com as linguagens e programas

As linguagens e conceitos utilizados nesta documentação eram-me desconhecidos pelo que antes de começar a interagir com os programas de teste e a modificar conteúdo, tive que estudar alguns conceitos, até porque um dos conteúdos que tive que desenvolver para a documentação era um género de introdução que

explicava como a mesma funcionava e não conseguia fazer isso sem perceber os conceitos. Esta introdução pode ser encontrada no website na secção *Get Started* e é, em muito, parecida, com o que passo a explicar a seguir **(Fig.56)**.

Tanto para os utilizadores como para mim, é necessária a instalação de uma aplicação de testes de Web Services, que no meu caso foi o **SoapUI**.

Para testar os serviços neste programa e perceber como ele funcionava, era necessário ter conhecimentos mínimos sobre os conceitos de SOAP, WSDL, REST e URI.

Após algum estudo consegui perceber que:

- **SOAP - Pedidos:**
  - Os pedidos feitos feitos em SOAP têm um esqueleto específico que é sempre constituído por: *envelope, header* e *body*. A tag *envelope* está sempre presente, tanto no pedido como na resposta, e é o que envolve/engloba o *header* e *body*. O *envelope* é responsável por identificar o pedido como uma mensagem SOAP.
  - A tag *header* também está sempre presente no pedido. Muitos serviços desta documentação, requerem autenticação por parte do utilizador e, quando isto acontece, as credenciais de autenticação são enviadas através do *header*. Quando não é necessária a autenticação, a tag continua a estar lá, simplesmente não tem dados.
  - A tag *body* também está sempre presente, mas ao contrário do *header*, a tag *body* tem sempre que conter algum comando que, pode ou não conter parâmetros, porque o *body* é o elemento que contém as informações de chamada e resposta. Isto significa, que mesmo que o *header* esteja vazio, o *body* irá sempre ter algo para solicitar e responder.
  - No caso da autenticação do serviço ser mandatória, é obrigatório o *header* ter as credenciais, pois a informação solicitada no *body* vai depender da informação do cliente que está a ser autenticado.

- **SOAP – Respostas:**
  - Como foi dito no ponto anterior, o *envolope* está sempre presente, seja nos pedidos ou nas respostas.
  - Ao contrário do que acontece nos pedidos, nas respostas o *header* nunca aparece.
  - Só é devolvido o *body* com a informação pedida no seu contéudo.
  - Em caso de erro durante o pedido, o *body* não irá retornar o que foi solicitado e, em vez disso, irá mostrar a tag *fault* que é o elemento responsável por armazenar os erros e os estados da informação. Sempre que o *fault* aparece, é mostrado como um elemento *child* do *body*, isto é, a tag *fault* aparece entre as tags *body*.
- **REST – Pedidos:**
  - Em REST, temos o *header* e o *body.*
  - Ao *header*, sempre que é necessária autenticação, acrescem os seguintes campos: *Username, Password, Nonce* e *Created.*
  - Quando é necessário adicionarmos parâmetros e o REST está a utilizar o método GET, os parâmetros podem ser incluídos no *header* (Exemplo – Key: value) ou podem ir diretamente no URI (exemplo - <HOST>/rs/path/getFunction**?key=value**).
  - Caso esteja a ser utilizado o método POST, os parâmetros têm que ser introduzidos no *body* que vem a seguir ao *header* e é estruturado da seguinte maneira:

    {

    “key_1”:“value_1”,

    “key_2”: “value_2”

    }

- **REST – Respostas:**
  - A resposta do REST tem a mesma estrutura que o *body*. A primeira linha contém o nome da função a ser chamada, na segunda linha temos o *return* e dentro do *return* temos os valores returnados segundo a função e parâmetros pedidos:

{“getFunctionResponse”:
{“return”:
{
“key_1”: “value_1”,
“key_2”: “value_2”,
“key_3”: “value_3”,
“key_4”: “value_4”
}
}
}

Após ter compreendido como funcionavam estas tecnologias, consegui então começar a testar o código que se encontra na documentação e verificar se todos os códigos que lá eram apresentados estavam corretos e devolviam sempre informação sem erros.

Como mencionei antes, os testes foram realizados no programa SoapUI. Para começarmos, temos que abrir o programa e criar um novo projeto em SOAP ou em REST **(Fig.57)**. Maior parte dos projetos que testei foram em SOAP. Ao consultar os serviços disponíveis nas seções do menu da documentação, podemos observar que na coluna central, depois do título de cada serviço, existe sempre um lembrete a avisar a autenticação é ou não obrigatória nesse respetivo serviço. Se passarmos à coluna da direita, onde são mostrados os códigos, iremos encontrar antes do código o WSDL se tivermos no separador e o URI se tivermos no separador REST **(Fig.58)**.

Após criarmos o projeto no SoapUI, precisamos de copiar o WSDL, caso tenhamos criado um projeto SOAP e o URI caso tenhamos criado um projeto REST **(Fig.59)**. Depois de copiar um destes links colamos num campo específico no SoapUI que irá, agora, carregar todos os serviços disponíveis para esse link, para o projeto criado e podemos então testá-los **(Fig.60)**.

Existem dois tipos de links, que fazem parte tanto do WSDL como de URI e, isto acontece, porque existem dois tipos de ambiente nos quais podemos testar o código: ambiente de integração e ambiente produção. O ambiente de integração serve para a equipa testar o código livremente, num ambiente que imita o ambiente onde irá estar o produto final, mas ao qual só nós temos acesso. Enquanto o ambiente de produção enquadra-se numa etapa final onde os serviços já estão disponíveis igualmente para todos e não só para nós (equipa).

Os links representam-se da seguinte forma:

- **Ambiente de integração:** https://**i**mobics.mobiag.com/mobics-webservices/...
- **Ambiente de produção:** https://mobics.mobiag.com/mobics-webservices/...

Como podemos observar, a única diferença, neste caso, é a letra "i" depois de "https://" e antes de "mobics", no ambiente de integração. Estes links, constituem sempre a parte inicial do WSDL e do URI.

Agora, um exemplo de WSDL e URI, em ambiente de integração:

- **WSDL:** https://imobics.mobiag.com/mobics-webservices/**CarClub?wsdl**
- **URI:** https://imobics.mobiag.com/mobics-webservices/**rs/carclub/getCustomerCarClub**

É possível observar que as partes iniciais do WSDL e do URI são as dos links mostrados anteriormente e que, a parte final do link indica o serviço que queremos carregar para o SoapUI.

Neste caso, o WSDL e o URI irão transferir para o projeto o serviço que diz respeito ao *Customer Car Club*. Se consultarmos a documentação, ficamos a saber que este serviço requer autenticação e que não tem nenhum *input* **(Fig.58)**. Desta maneira, no caso do SOAP, simplesmente introduzimos as credenciais no *header* e clicamos *Enter* e será devolvida toda a informação sobre o *Car Club* daquele cliente, que sabemos quem é, através dos dados de autenticação **(Fig.62)**.

4.4 Organização e edição do código

Depois da etapa anterior, chegou a altura de dedicar-me a aprender a nova sintaxe do Markdown para que pudesse passar a alterar o código e criar novo conteúdo. Esta foi a parte mais fácil, pois a sintaxe Markdown é bastante fácil e demorei pouco tempo até aprendê-la.

Depois ter efetuado os testes ao código da documentação, obtive alguns erros que eram necessários corrigir e isto era feito no código, através do Sublime, que foi referido no início deste capítulo.

O primeiro problema com que me deparei é que todo o conteúdo do menu, encontrava-se num só ficheiro, o que acabava por ser quase 8000 linhas de código e cada vez que precisava de encontrar um tópico tinha que fazer scroll durante bastante tempo ou usar a função *Find* do Sublime mas, de qualquer maneira, era demasiado moroso e trabalhoso. Posto isto, fui consultar a documentação do Slate para saber se era possível usar *includes* na documentação e, caso fosse, como o fazia. A documentação do Slate tinha a solução e, passa por criar uma pasta no servidor com o nome *includes* e criar lá dentro um ficheiro novo para cada tópico do menu e chamá-los a todos, pela ordem que queremos que apareça no website, no ficheiro principal (index.html.md) **(Fig.63)**. Posteriormente, sempre que quisesse editar uma determinada seção do menu, era só navegar até ao ficheiro através da palete de comandos do Sublime e abri-lo e editá-lo (Github – *Includes*) **(Fig.64 e 65)**.

O segundo aspeto, que não pode ser definido como um problema, mas que me fez consultar também a documentação do Slate, foi o facto de querer adicionar um favicon ao website com o logótipo da Mobiag. A solução encontrada na documentação, foi bastante simples. O primeiro passo foi criar o favicon e colocá-lo na pasta *images* do servidor, o que foi feito através do Filezilla, pois no Sublime não descobri como fazer *upload* de imagens. O segundo passo, foi aceder através da palete de comandos do Sublime, à pasta usual do servidor, onde se encontram todos os ficheiros da documentação e entrar na pasta *layouts* e abrir o ficheiro *layout.erb*. Neste ficheiro, tive que colar a seguinte linha de código na seção *head*: **<%= favicon_tag 'images/favicon.ico' %>** e o favicon passou a aparecer no website (Github - *Favicon*) **(Fig.66)**.

Quando comecei a mexer na documentação, as seções existentes no menu eram as seguintes: *Introduction, Get Started* (sem conteúdo, só o título), *Authentication, Search Cars, Booking a Car, Report Damage, Operator* e *Subscriptions.* Durante o meu estágio, fiquei responsável por criar o contéudo para o *Get Started*, como referi anteriormente e, ainda acrescentei à documentação as seguintes seções: *Customer, Tags, Password Recovery* e melhorei o conteúdo e apresentação das existentes. O conteúdo para estas novas seções foi retirado da Wiki, uma plataforma da Atlassian que a Mobiag utilizava que permitia que todos os membros da equipa com acesso ao website pudessem adicionar e editar conteúdo sobre o que tinham feito. Neste caso, eram os programadores que adicionavam os tópicos das novas versões que eram lançadas e a explicar como funcionava e eu punha essa nova informação na documentação da API.

A base da estrutura das seções era a seguinte:

- Título com o nome da seção.
- Breve explicação do que o serviço selecionado faz.
- Uma barra informativa a avisar se a autenticação era obrigatória ou não naquele serviço.
- Tabela de *inputs*, ou seja, os parâmetros que eram necessários inserir na mensagem SOAP ou REST. Caso não houvesse *inputs,* também não havia tabela. Simplesmente uma frase a dizer que aquele serviço não tinha *inputs*.
- Tabela de *outputs*, ou seja, depois de testar o código, quais são os parâmetros que é suposto o serviço devolver. Também pode não haver *outputs* e, nesse caso, deixa de existir tabela.
- Do lado direito, o código SOAP ou REST.

CONCLUSÃO

No plano de estágio que foi necessário entregar antes de iniciar o estágio estavam estipuladas determinadas tarefas que iria estar encarregue durante o estágio. Essas tarefas eram:

1. Testes de usabilidade (UX/UI)
2. *Mockups* de Interfaces
3. Análise e testes de APIs
4. Utilização de Web Services
5. Testes e documentação de código
6. Revisão e melhoramento de código

Durante o estágio tentei ao máximo dar o meu melhor e tentei aprender o mais que podia e ganhar experiência, tanto na minha área como em assuntos que se tornam necessários aprender no mundo profissional.

Das tarefas estipuladas, consegui realizar todas exceto o *mockup* de interfaces. Isto não deixou de acontecer pela minha falta de vontade mas sim por razões que dizem respeito à Mobiag e à Citydrive. Inicialmente, era suposto eu realizar o teste de usabilidade à aplicação e, de seguida, com os problemas encontrados, fazer o redesign da aplicação, através da elaboração de *mockups* de interfaces, *wireframes*, *walktroughs* e testes de usabilidade. Este processo ia resultar numa aplicação com uma experiência de utilização muito melhor que a atual, que na minha opinião iria fazer com que mais pessoas utilizassem a aplicação. Além de não poder ter seguido em frente com este processo, também havia sempre o factor condicionante que a aplicação já estava praticamente toda desenvolvida, o que significa que se eu mudasse por completo o design da aplicação implicava também que os programadores tivessem que voltar a aplicação do início, o que estava fora de questão. A ideia, então, seria pegar na estrutura e design da aplicação existente e tentar redesenhá-la o melhor que pudesse para aumentar o nível de usabilidade da mesma.

Independentemente do facto de não poder ter realizado esta parte do projeto, acredito que fiz um bom trabalho ao identificar os erros técnicos e de usabilidade mais graves que eles possuíam na aplicação. Durante a minha licenciatura, tive Design de

Interação, que abordou tópicos como o *UX/UI Design* que consegui agora aprofundar e compreender melhor com a realização deste estudo de usabilidade. Durante este mestrado, também tive a disciplina de Standards de Usabilidade e Acessibilidade que também veio a ser útil durante o estágio, no sentido que me ajudou a distinguir os conceitos de usabilidade e design de experiências, que muitas vezes se confundem e são interpretados como sendo o mesmo conceito.

Em relação às outras tarefas, estavam relacionadas com uma área com a qual não estava familiarizada e não tinha conhecimentos sobre os mesmos, o que tornou todo o percurso num desafio. Para mim, foi a melhor parte do estágio, pois permitiu-me aprender uma nova sintaxe (Markdown), a compreender o que é uma API e o que são *Web Services*, o que fazem e como funcionam. A partir destes dois últimos, aprendi também a conseguir ler minimamente a linguagem XML e compreendi como funcionam as mensagens SOAP e REST. O XML, SOAP e REST, foram os conceitos que levei mais tempo a perceber como funcionavam e como interagiam com os *Web Services*, porque para mim tudo o que se passava era demasiado abstrato e tinha dificuldade em imaginar o que se estava a passar por trás de tudo. Após o estágio, não tive mais contacto com estas tecnologias e não é uma área que me interesse seguir, mas posso dizer que consigo perceber e explicar minimamente os conceitos, o que é sempre uma mais-valia para uma pessoa que trabalha na minha área (*Design, Web Design, Frontend*).

No que diz respeito à documentação, estou orgulhosa por ter estado responsável or um projeto deste género, porque confiaram-me a documentação, mesmo sabendo que eu não era familiarizada com os conceitos e que o que eu ia realizar, ia ser apresentado a clientes deles. Conquistei todas as etapas que me foram propostas e as que me propus a fazer sozinha, como a situação de dividir todos os ficheiros em *includes* para ser mais fácil de aceder e utilizar o código.

Em relação à Mobiag, foi um grande privilégio trabalhar para eles. Receberam-me todos com grande hospitalidade e fizeram-me sentir parte da equipa durante todo o estágio. Estavam sempre prontos a ajudar em qualquer situação, profissional ou pessoal. Ganhei também experiência não só a realizar o meu trabalho, mas também a conviver com o resto da equipa que tinha sempre opiniões e críticas construtivas a dar. Aprendi que o trabalho em equipa faz com que os projetos se desenvolvam mais

rapidamente, pois tinha o hábito de tentar conseguir fazer tudo sozinha para ter um pouco de mérito próprio e, mais tarde, apercebi-me que podia resolver tudo mais rapidamente se tirasse dúvidas, por exemplo, com o meu orientador de estágio. Isto beneficiava toda a empresa, uma vez que a velocidade com que passava a realizar o meu trabalho era maior enquanto antes perdia bastante tempo a pesquisar as minhas dúvidas e, muitas vezes, não conseguia chegar a lado nenhum e era obrigada a ter que me dirigir a alguém da equipa de qualquer maneira.

Além da Mobiag, existia no mesmo espaço a empresa Logistema, os quais também me fizeram sentir muito bem recebida e sempre me tentaram ajudar com tudo o que precisei, mesmo não sendo da mesma empresa que eles. Todo este ambiente, ajudou-me a ganhar motivação e a manter o meu ritmo de trabalho ao longo de todo o estágio.

Em retrospectiva, apesar de não ter realizado certas tarefas e ter trabalhado em projetos mais relacionados com a minha área, considero que este estágio, de uma forma geral, foi bastante bom e importante para mim a todos os níveis, seja profissional ou pessoal, pois permitiu-me amadurecer certos aspetos que me irão ajudar no meu futuro. Considero que obtive bons resultados e acredito que a Mobiag tenha gostado de me ter como sua estagiária.

BIBLIOGRAFIA(S) / REFERÊNCIAS BIBLIOGRÁFICAS

- Brito, Francisco (2016) – Relatório de estágio de mestrado em Novos Media e Práticas Web: *Usabilidade e Design de Interfaces para uma aplicação Web da Premium Minds*.

- Delgado, Carla (2015) – Relatório de estágio de mestrado em Novos Media e Práticas Web: *A importância dos testes de usabilidade e do Responsive Web Design no desenvolvimento de projetos digitais.*

- Mobiag – *About us* (2017). Disponível em: http://www.mobiag.com/en/about

- Mobiag – *Clients* (2017). Disponível em: http://mobiag.com/en/clients

- Mobiag – *Mobiag Solution: App & Webportal* (2017). Disponível em: http://mobiag.com/en/mobiag-solution/app-webportal

- Mobiag – *Mobiag Solution: mobiCS* (2017). Disponível em: http://mobiag.com/en/mobiag-solution/mobics

- Mobiag – *Mobiag Solution: Mobiag Connect* (2017). Disponível em: http://mobiag.com/en/mobiag-solution/mobiag-connect

- Mobiag – *Mobiag Solution: App & Webportal* (2017). Disponível em: http://mobiag.com/en/mobiag-solution/mobiag-network

- Wikipédia – *Carsharing* (2014). Disponível em: https://en.wikipedia.org/wiki/Carsharing#Description

- Koivunen, Marja; May, Matt (2002) – *Exploring Usability Enhancements in W3C Process.* Disponível em: https://www.w3.org/2002/09/usabilityws.html

- Nielsen, Jakob (2012). Disponível em: https://www.nngroup.com/articles/usability-101-introduction-to-usability/

- Norman, Donald; Nielsen, Jakob (2010) - *Gestural Interfaces: A Step Backwards In Usability*. Disponível em: http://www.jnd.org/dn.mss/gestural_interfaces_a_step_backwards_in_usability_6.html

- Morville, Peter (2004) - *User Experience Design*. Disponível em: https://www.nngroup.com/articles/usability--101--introduction--to--usability/

- Allen, Jesmond; Chudley, James (2012*): Smashing UX Design Foundations for Designing Online User Experiences.* Chichester: Wiley;
- Weinschenk, Susan (2011): *100 Things Every Designer Needs to Know about People*. Berkeley, CA: New Riders;
- Cybis, Walter (2003) - *Engenharia de Usabilidade: uma abordagem ergonômica*. Disponível em: http://www.labiutil.inf.ufsc.br/hiperdocumento/unidade3_3_2.html
- Nielsen, J.; Molich, R. (1990) - *Heuristic Evaluation of User Interface.* Proc. ACM CHI'90 Conf. Seattle, WA, 1–5 April, 249-256.
- Nielsen, Jakob (1995) – *10 Usability Heuristics for User Interface Design.* Disponível em: https://www.nngroup.com/articles/ten-usability-heuristics/
- AA.VV. (n.d.) *User Interface Design Basics*. Disponível em: https://www.usability.gov/what--and--why/user--interface--design.html
- Lamprecht, Emil (2017) – *The difference between UX and UI design – A Layman's Guide.* Disponível em: https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/
- Wikipédia - https://en.wikipedia.org/wiki/Application_programming_interface#Web_APIs
- Facebook Developers - https://developers.facebook.com/docs/facebook-login
- W3Schools - https://www.w3schools.com/tags/ref_httpmethods.asp
- IETF Tools - https://tools.ietf.org/html/rfc2616
- W3Schools - https://www.w3schools.com/xml/xml_whatis.asp
- W3Schools - https://www.w3schools.com/xml/xml_tree.asp
- W3Schools - http://www.w3schools.com/xml/xml_services.asp
- Tutorials Point - https://www.tutorialspoint.com/webservices/what_are_web_services.htm
- W3Schools - https://www.w3schools.com/xml/xml_soap.asp
- Tutorials Point - https://www.tutorialspoint.com/restful/

- Tutorials Point - https://www.tutorialspoint.com/restful/restful_introduction.htm
- Sintaxe Markdown: Daring Fireball - https://daringfireball.net/projects/markdown/syntax
- Citydrive – *Sobre Nós* (2015). Disponível em: https://www.citydrive.pt/sobre-nos/
- Citydrive – *Como funciona* (2015). https://www.citydrive.pt/como-funciona/
- Sublime – *Installation.* Disponível em: https://wbond.net/sublime_packages/sftp/installation
- Sublime – *Usage.* Disponível em: https://wbond.net/sublime_packages/sftp/usage
- Documentação API Mobiag - http://api.mobiag.com:4567
- Github – *Documentation.* Disponível em: https://github.com/lord/slate
- Github – *Includes.* Disponível em: https://github.com/lord/slate/wiki/Using-Includes
- Github – *Favicon.* Disponível em: https://github.com/lord/slate/wiki/Adding-a-favicon

GLOSSÁRIO

API – Application Programming Interface.

HTML – HyperText Markup Language.

SFTP – Secure File Transfer Protocol.

XML – eXtensible Markup Language.

WSDL – Web Services Description Language.

SOAP – Simple Object Access Protocol.

REST – Representational State Transfer.

ANEXOS – CAPÍTULO 3

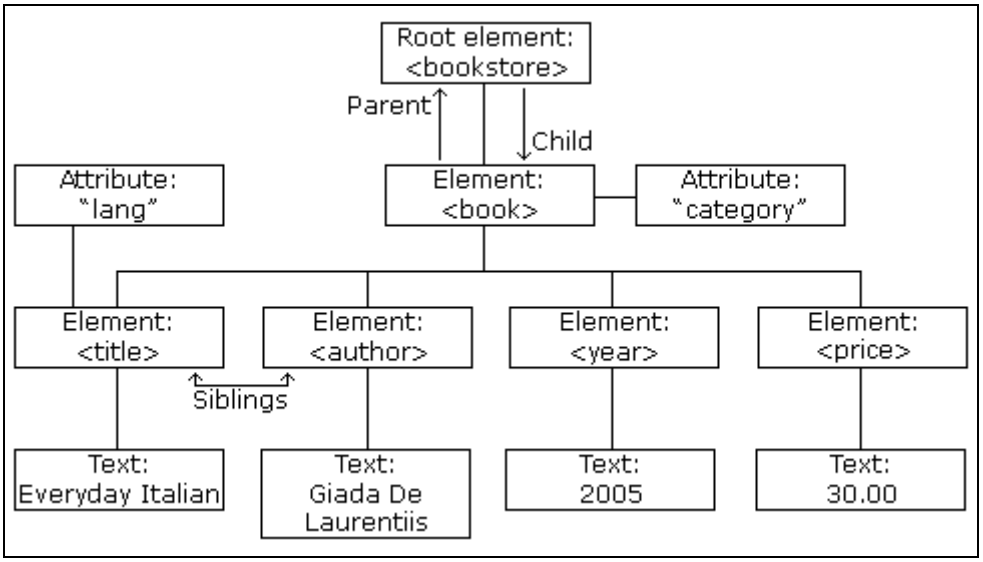


Figura 1 – *Node Tree* que explica a estrutura do XML.

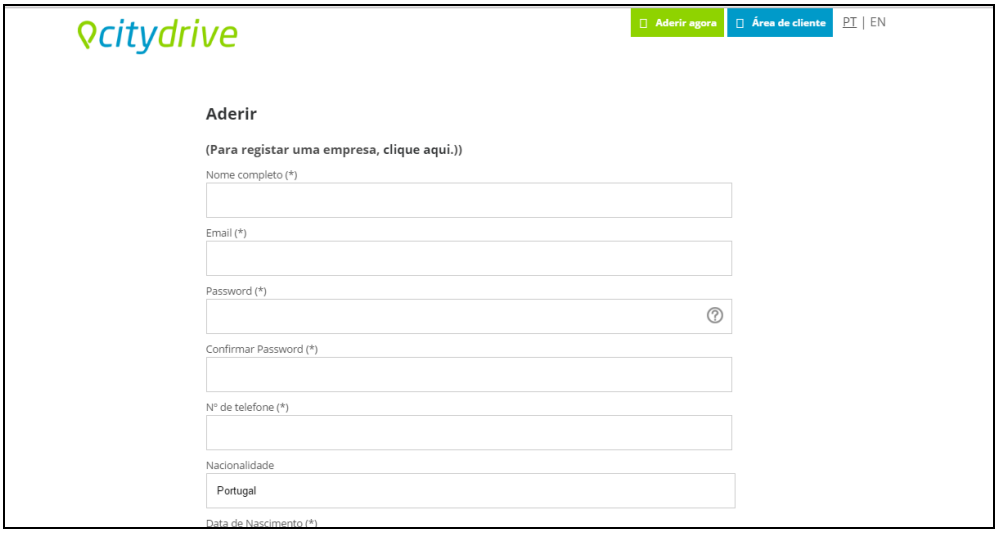


Figura 2 – Ecrã de Registo no website da Citydrive

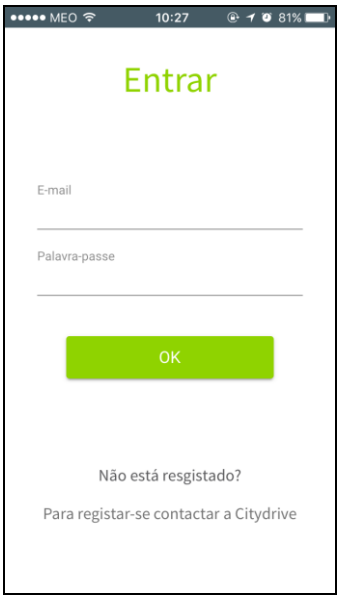


Figura 3 – Ecrã de login na aplicação Citydrive

(Continuação na página seguinte)

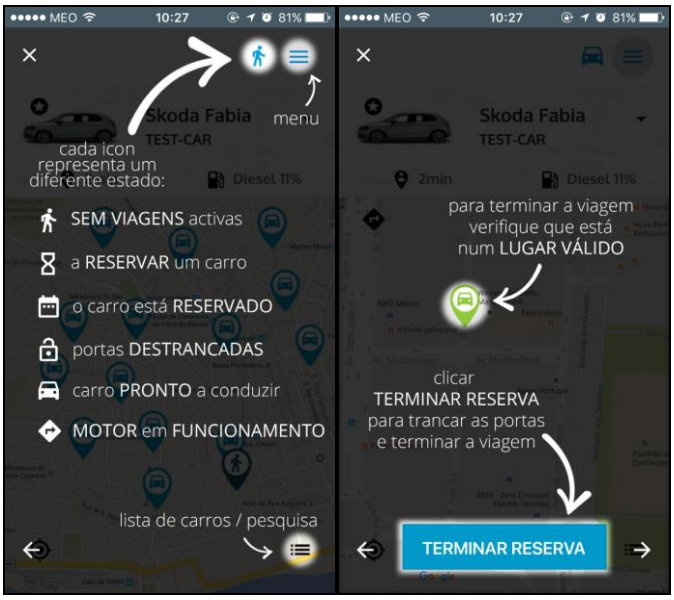


Figura 4 – Tutorial da aplicação Citydrive

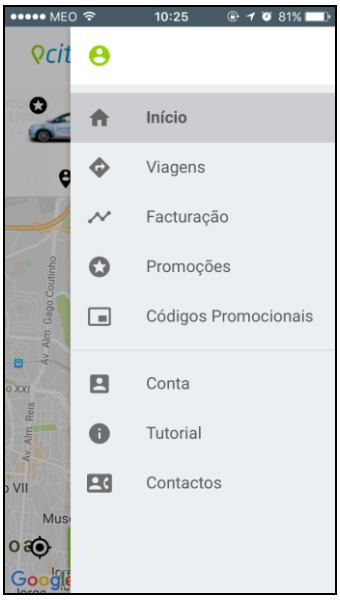


Figura 5 – Menu

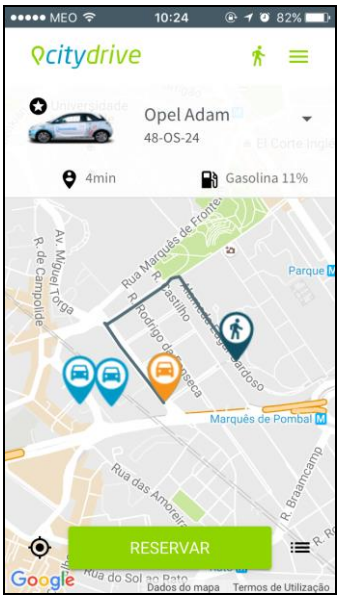


Figura 6 – Ecrã principal – mostra o carro mais próximo

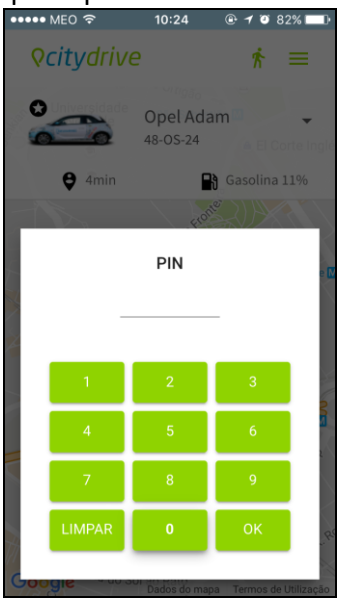


Figura 7 – Ecrã para inserir o PIN após clicar Reservar

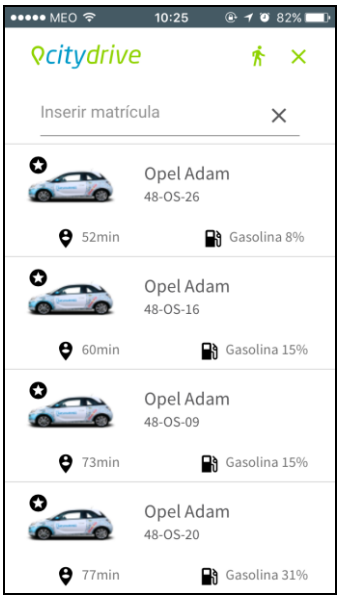


Figura 8 – Lista que aparece após clicar no botão pesquisa

Figura 9 – As setas do tutorial não são clicáveis

Figura 10 – Após clicarmos nos Termos de Utilização do mapa, não há como voltar para a aplicação

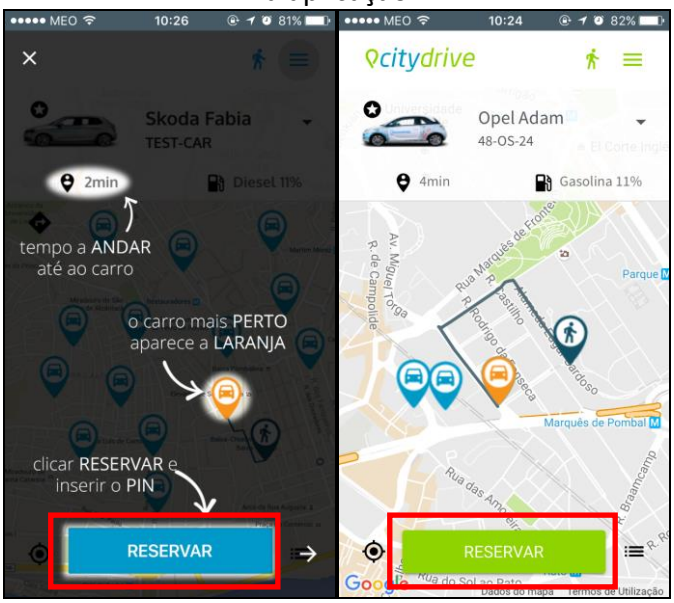


Figura 11 – Os botões Reservar não são consistentes na cor

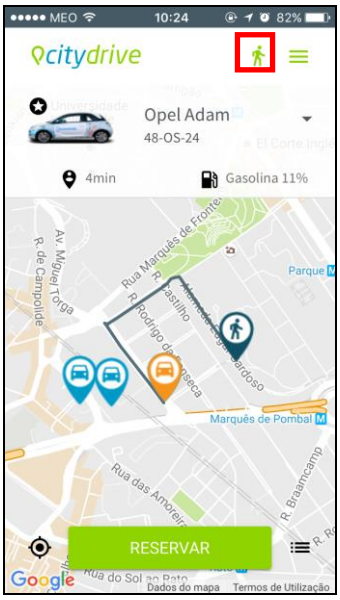


Figura 12 – Não é possível perceber o que significa o ícone "sem viagens ativas", se não virmos o tutorial.

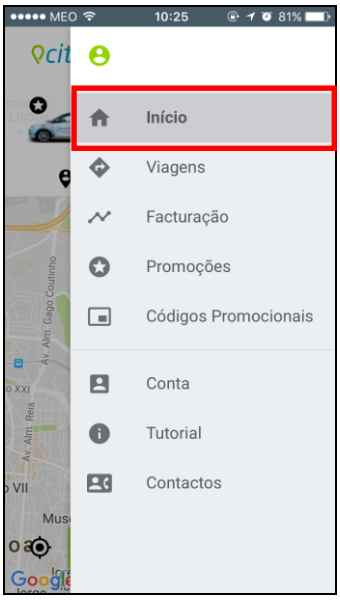


Figura 13 – Não existe um botão *Home* de acesso rápido. É preciso ir a Menu > Início.

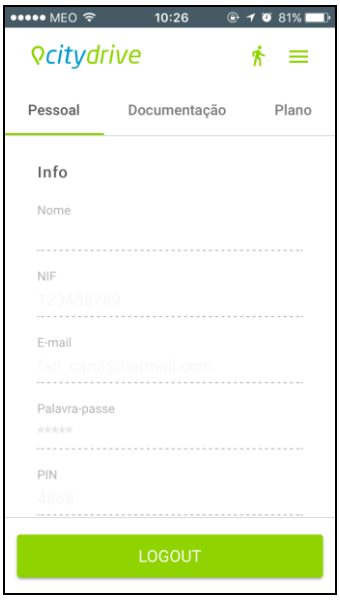


Figura 14 – Transmite ideia que é possível preenchermos os dados, mas não permite.

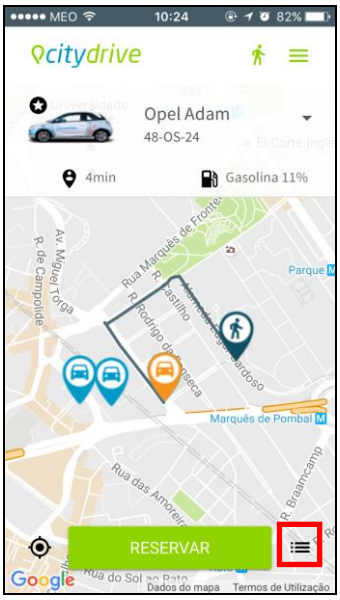


Figura 16 – O ícone de pesquisa de carros é pouco explícito e muitas vezes perde a visibilidade por causa do mapa.

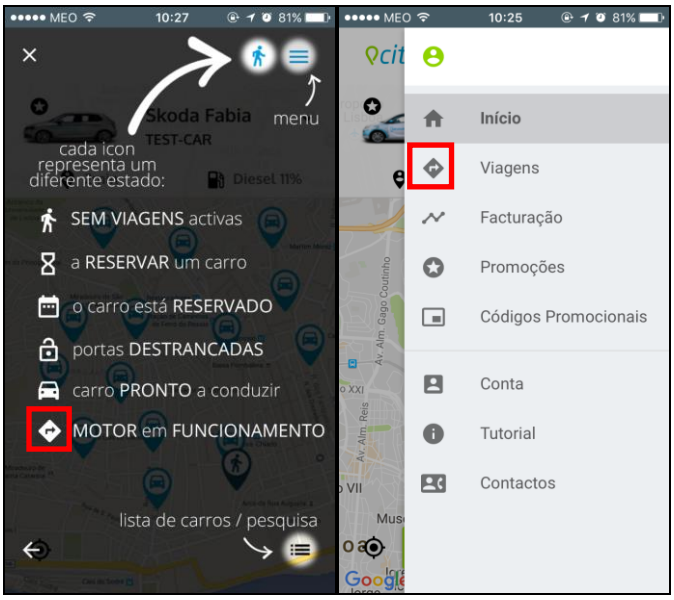


Figura 17 – O ícone que representa o motor em funcionamento é igual ao ícone que representa a seção Viagens.

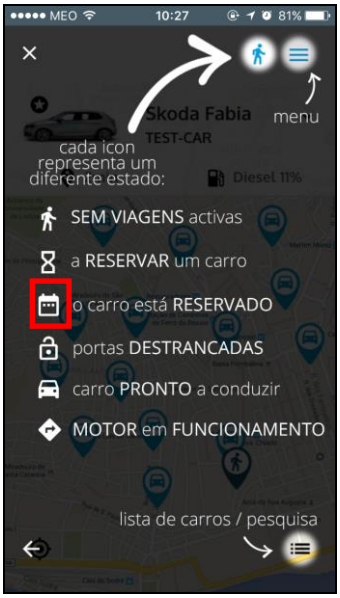


Figura 18 – O ícone de carro reservado não é percetível.

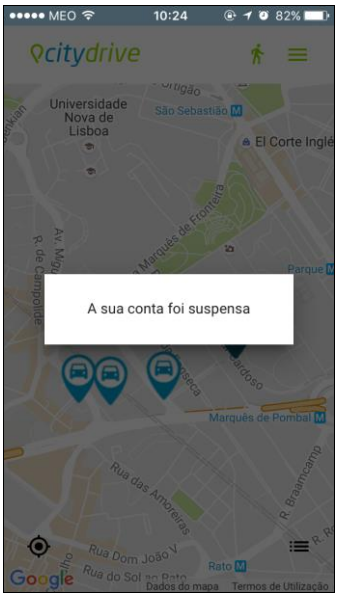


Figura 19 – Mensagem após tentarmos efetuar uma reserva sem saldo.

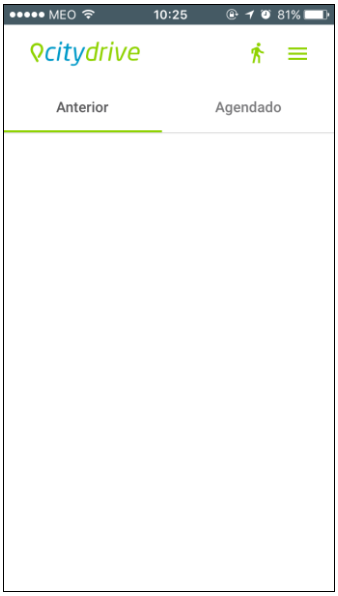


Figura 20 – Após acedermos a uma seção no Menu, não da para sabermos onde estamos, pois não existe um título.

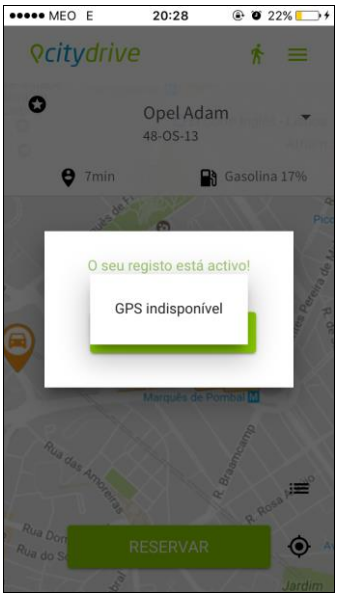


Figura 21 – Quando abrimos a aplicação sem termos o GPS ligado.

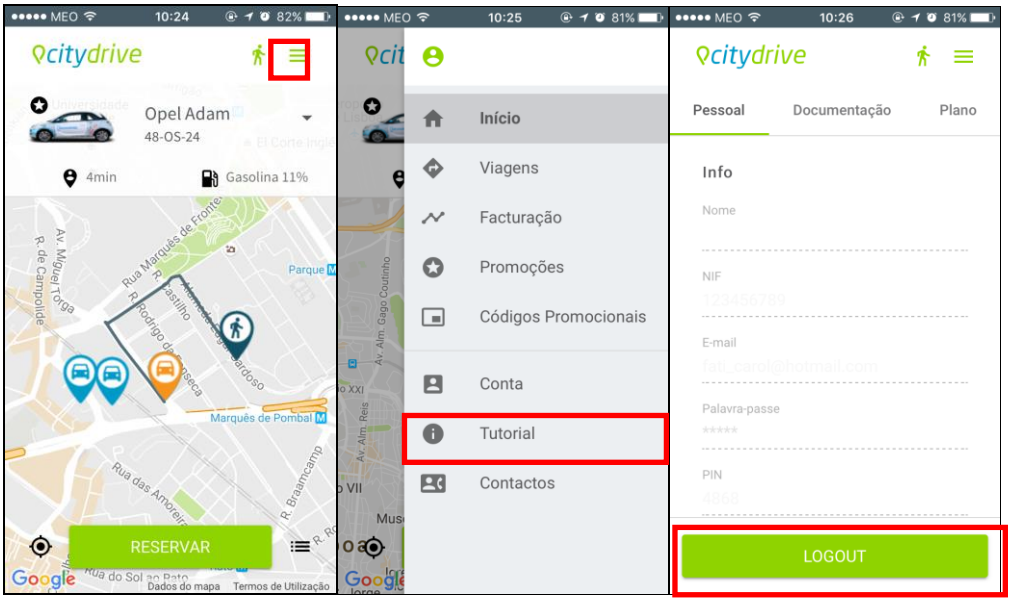


Figura 22 – São necessários muitos passos para fazer Logout.

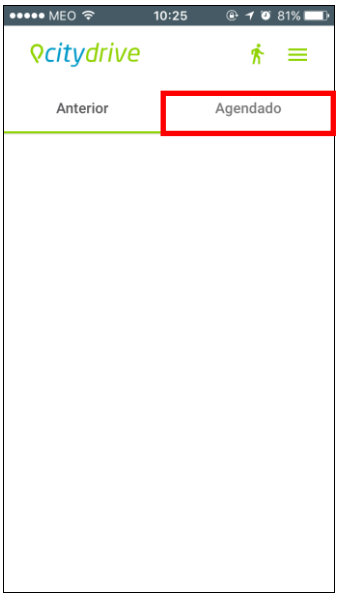


Figura 23 – Existe o separador Agendado, mas não é possível agendar viagens.

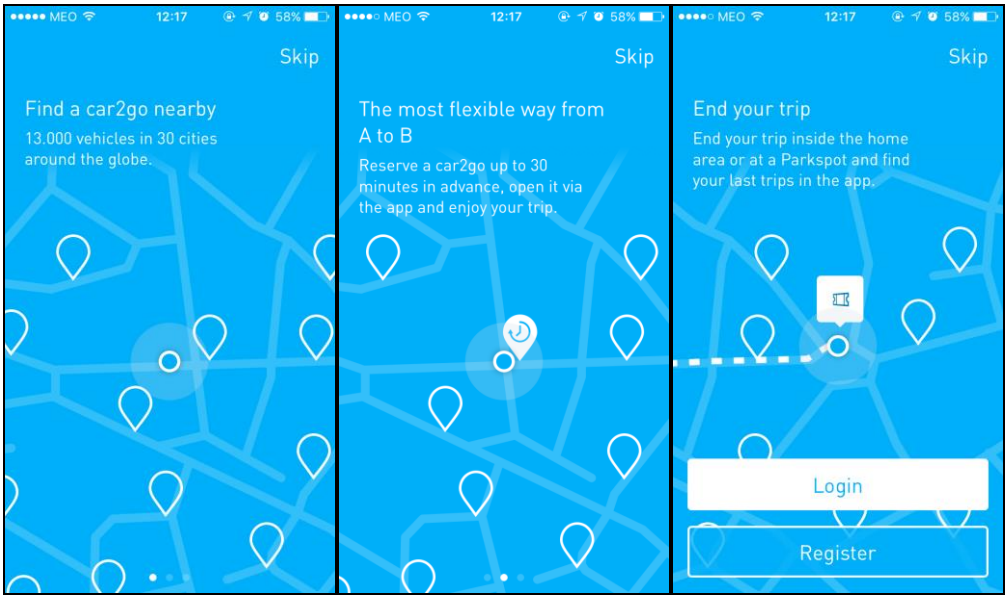


Figura 24 – Tutorial da aplicação *Car2Go*

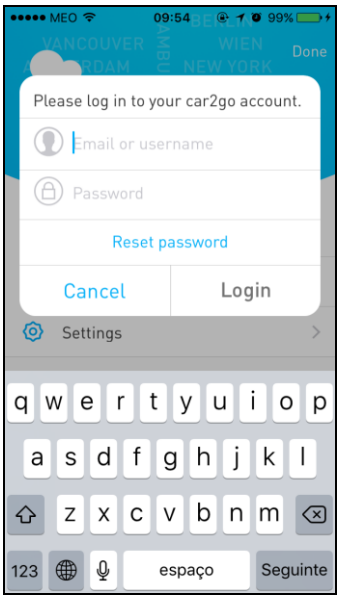


Figura 25 – Pop-up login da aplicação *Car2Go*

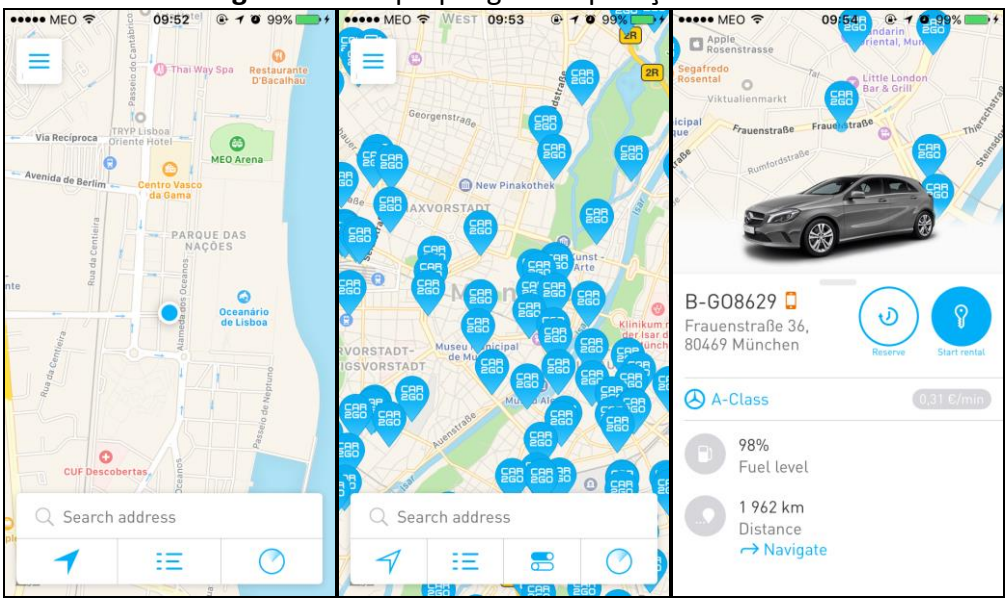


Figura 26 – Ecrãs principais da aplicação

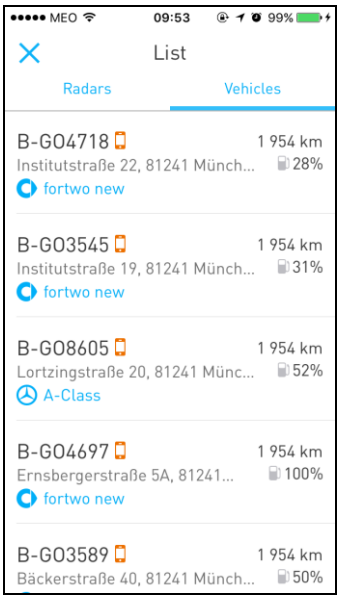


Figura 27 – Lista carros Radar

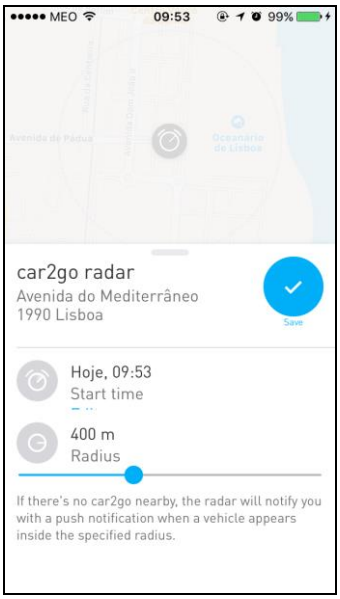


Figura 28 – Definir Radar

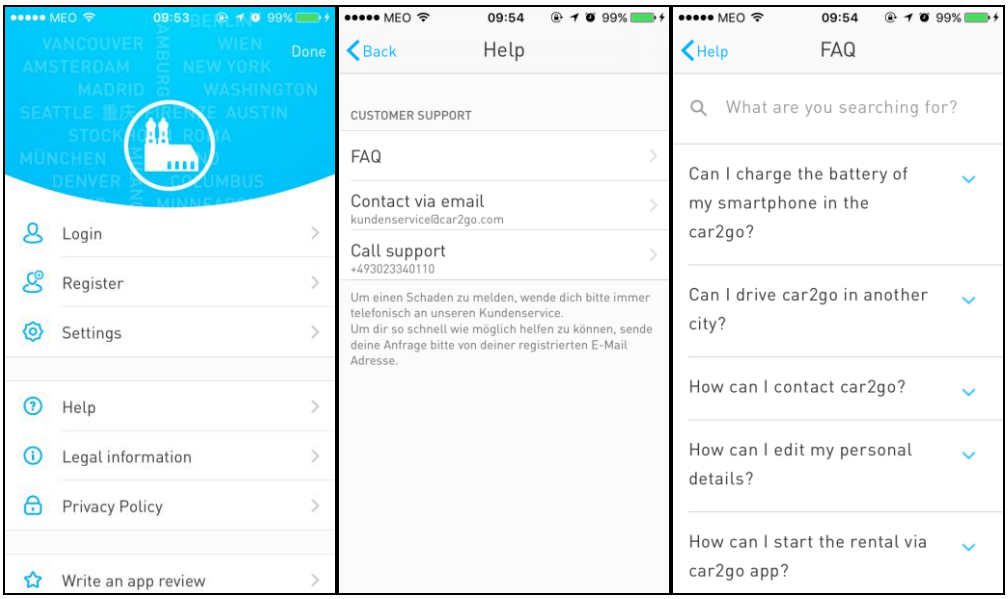


Figura 29 – Menu > Help

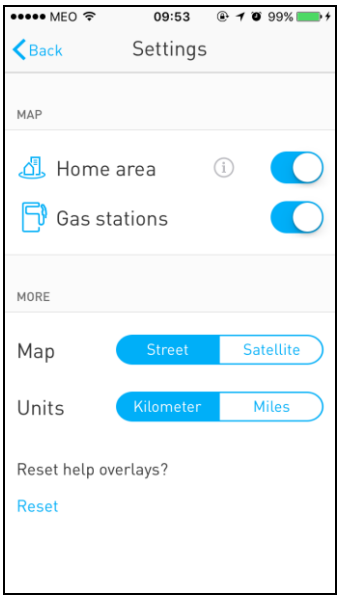


Figura 30 – Definições

Figura 31 – Ao fazer zoom out, mostra as cidades em que este serviço está disponível.

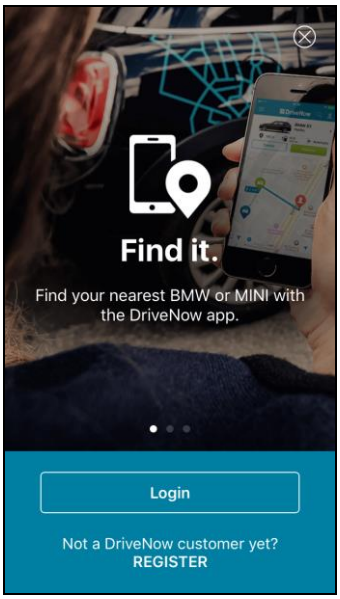


Figura 32 – Tutorial da aplicação *DriveNow*

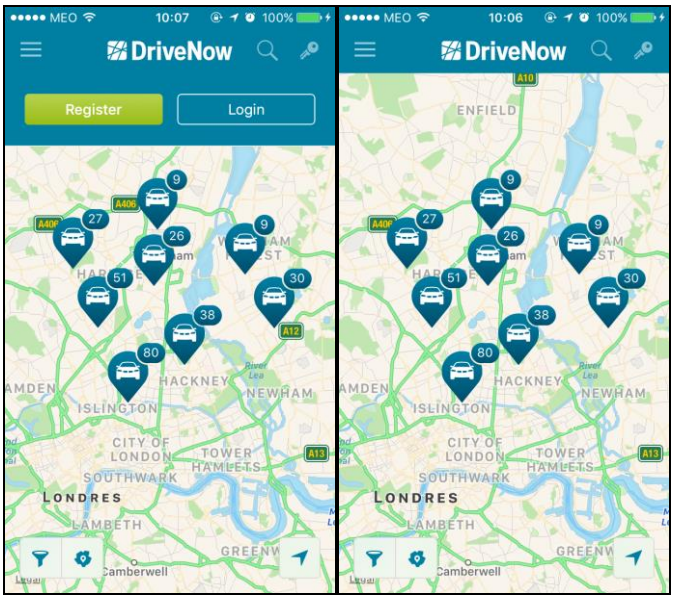


Figura 33 – Ecrã principal sem o login (esquerda) e com o login (direita).

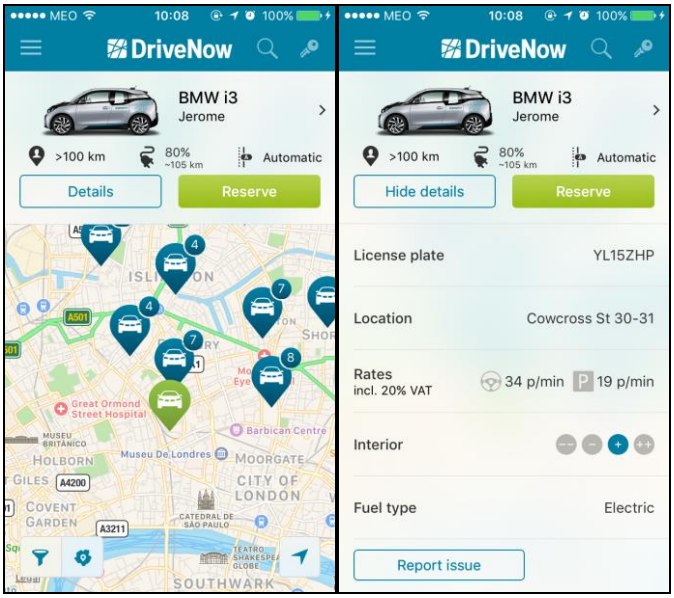


Figura 34 – O que acontece quando selecionamos um carro.

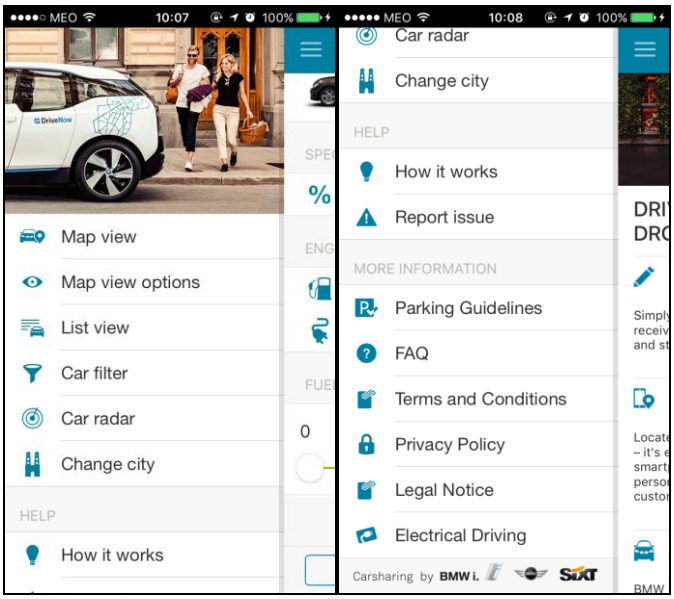


Figura 35 – Menu

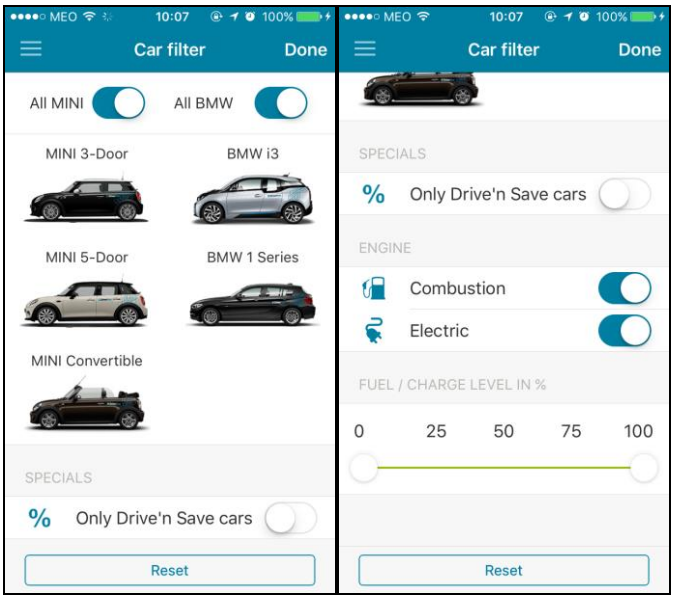


Figura 36 – Filtros de pesquisa de carros

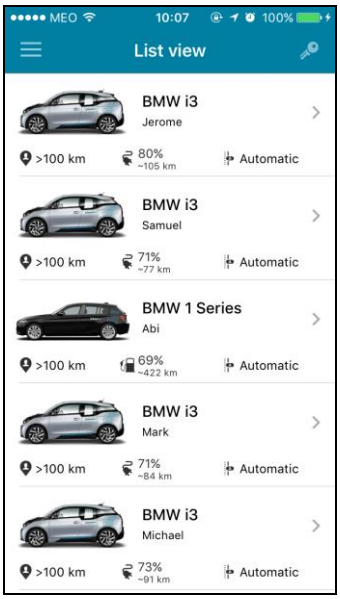


Figura 37 – Lista de carros disponíveis.

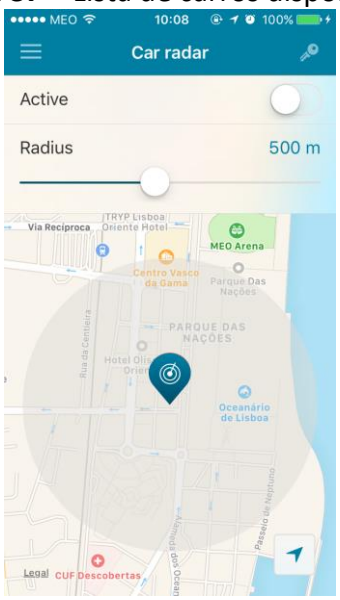


Figura 38 – Definir Radar.

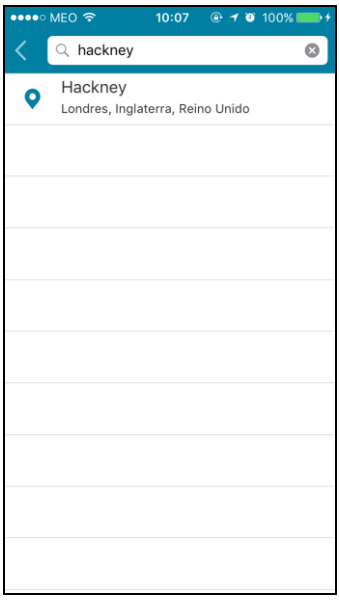


Figura 39 – Campo para inserir endereço.

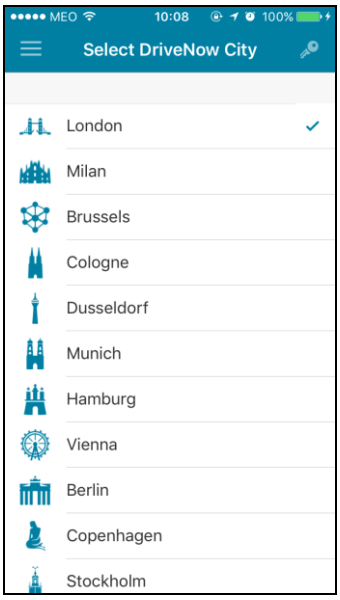


Figura 40 – Mudar a cidade.

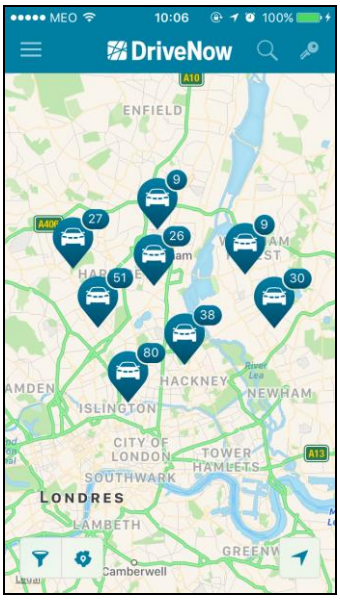


Figura 41 – Mostra o número de carros na zona, quando minimizado.

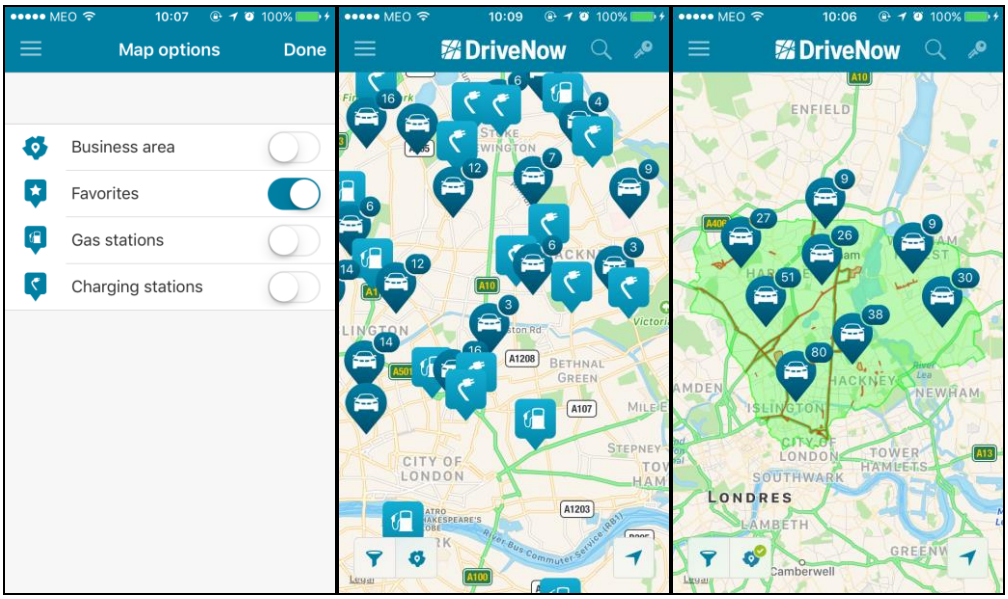


Figura 42 – Permite mostrar/ocultar a zona de serviço, favoritos, bombas de gasolina e postos de carregamento.

ANEXOS – CAPÍTULO 4

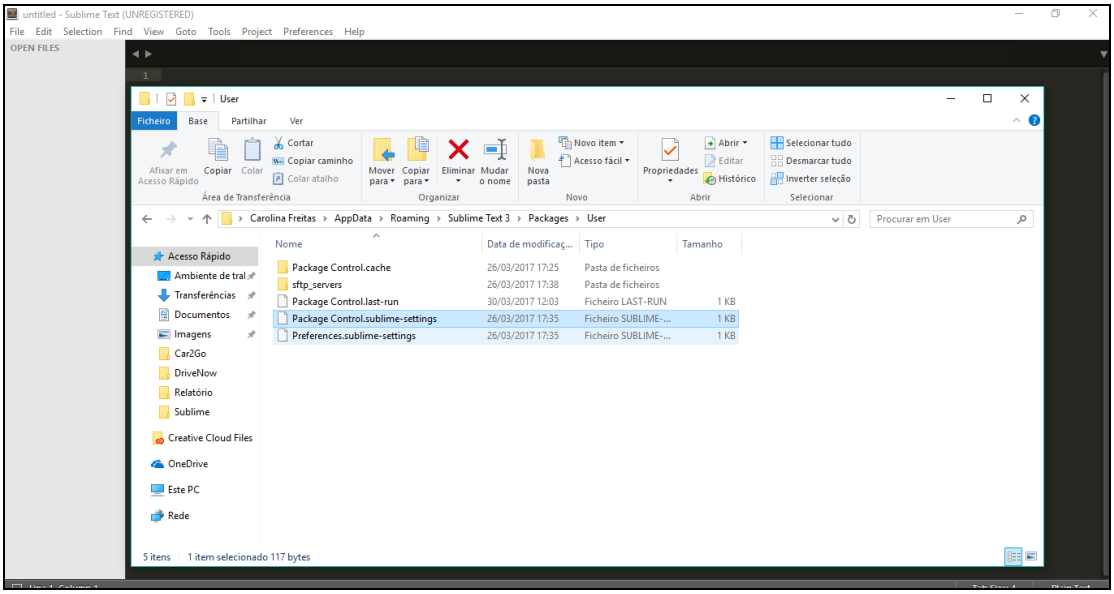


Figura 43 – Diretório do Sublime onde é colocado o *package.*

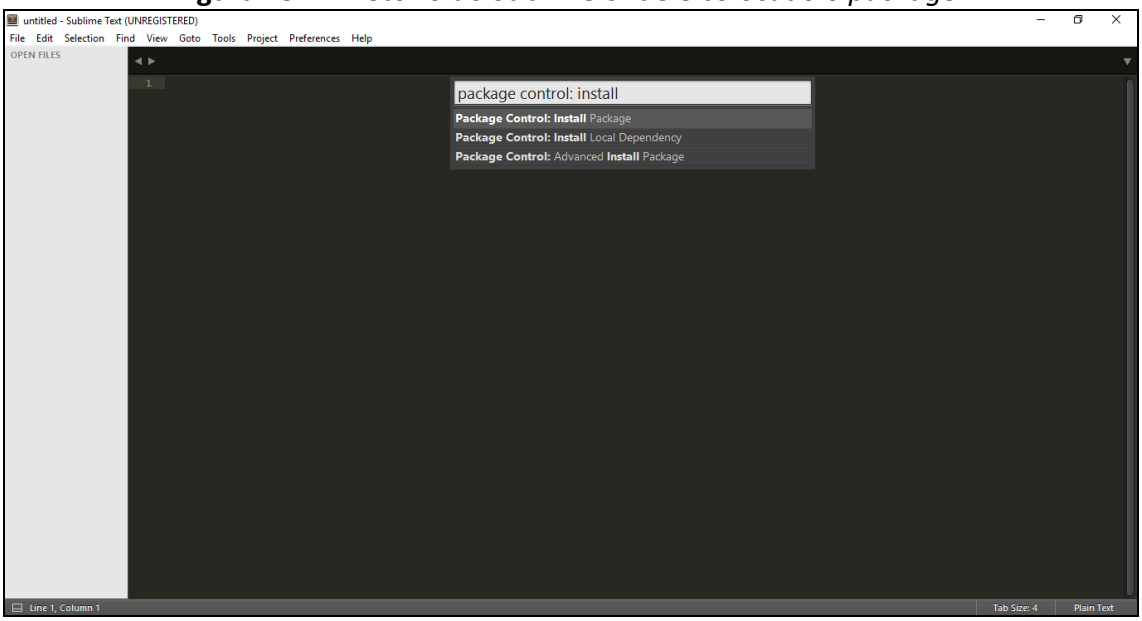


Figura 44 – Comando para instalação do *package.*

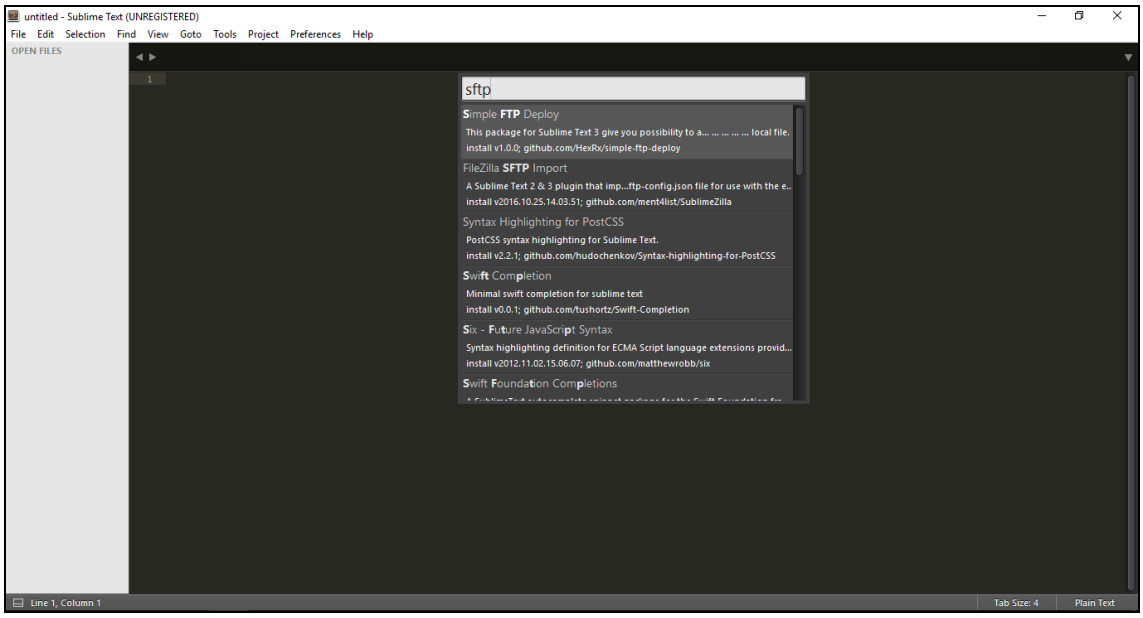


Figura 45 – Etapa onde temos que selecionar o SFTP Sublime para instalar.

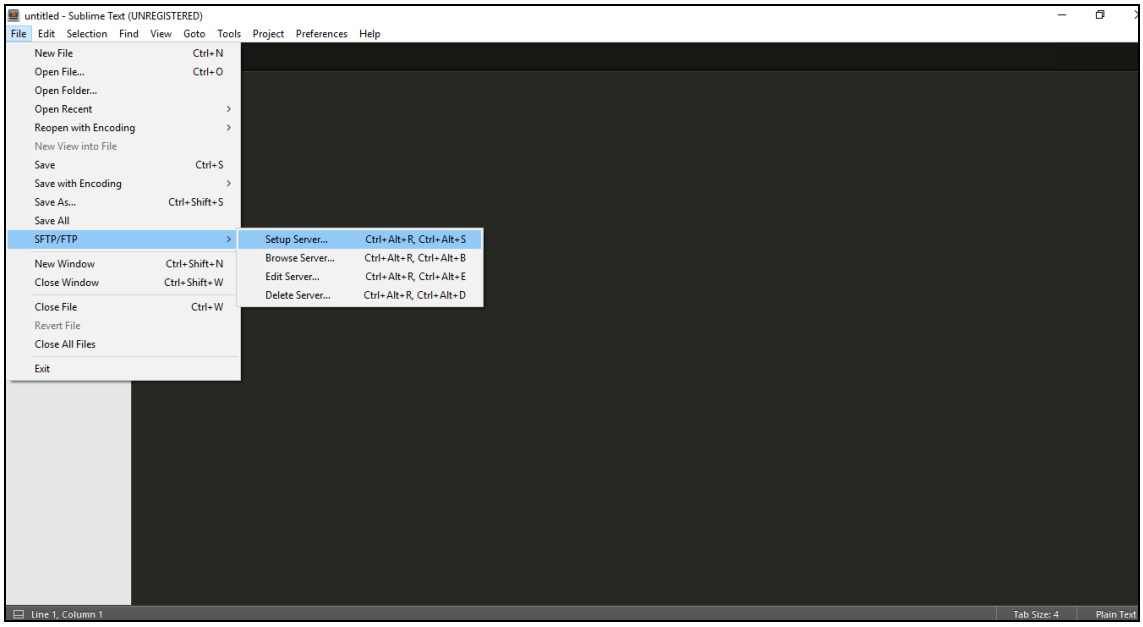


Figura 46 – Passos para configurar o servidor.

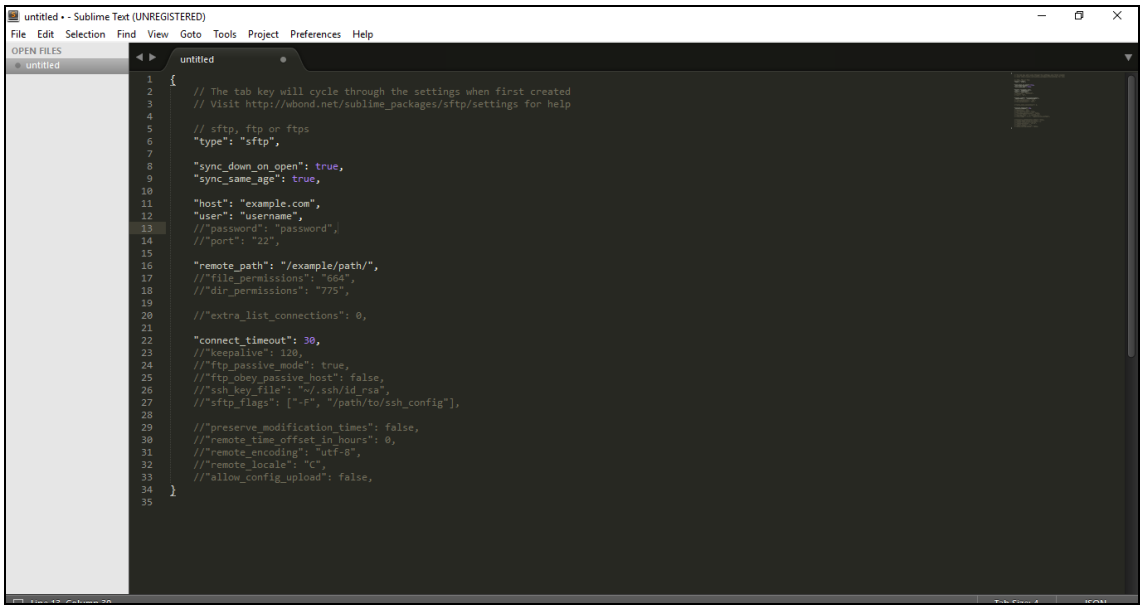


Figura 47 – Ficheiro para configurar o ficheiro com os dados do servidor.

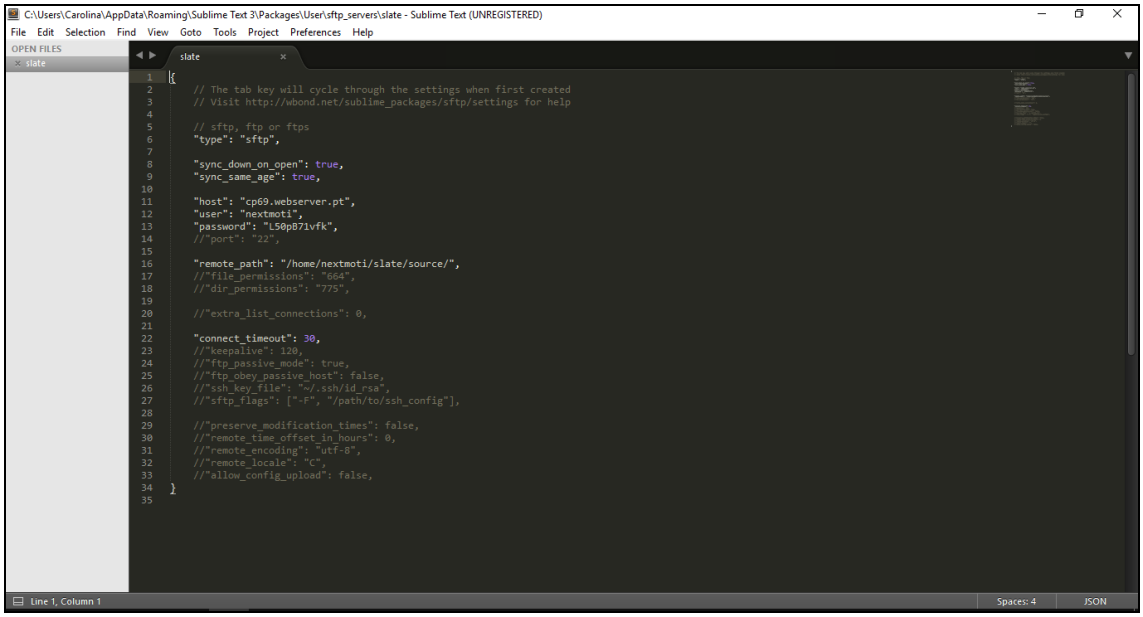


Figura 48 – Ficheiro já configurado.

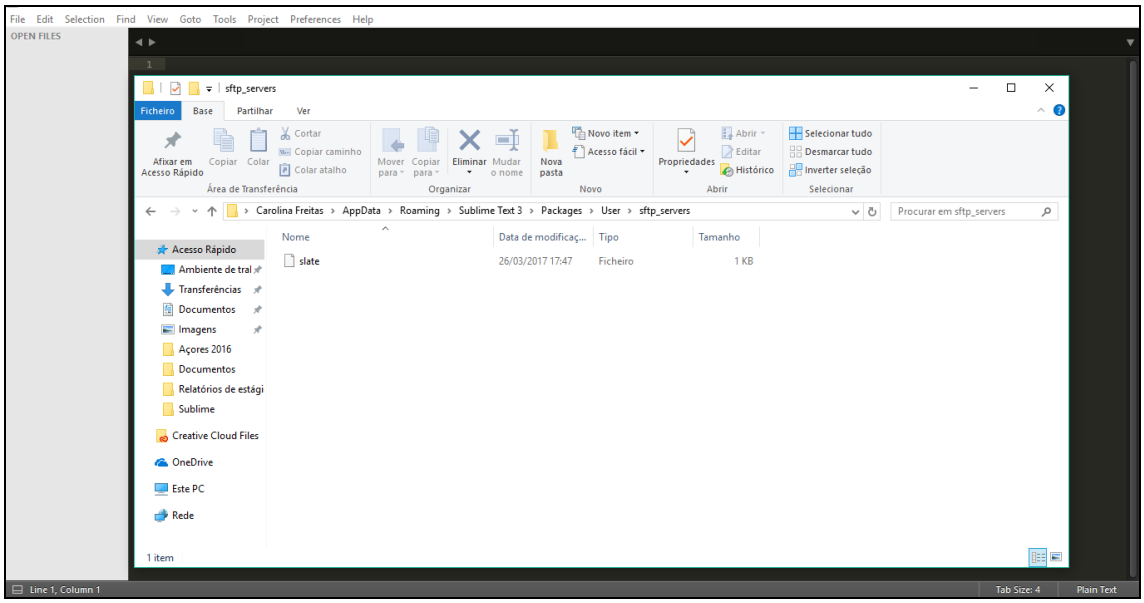


Figura 49 – Ficheiro guardado com o nome *Slate*.

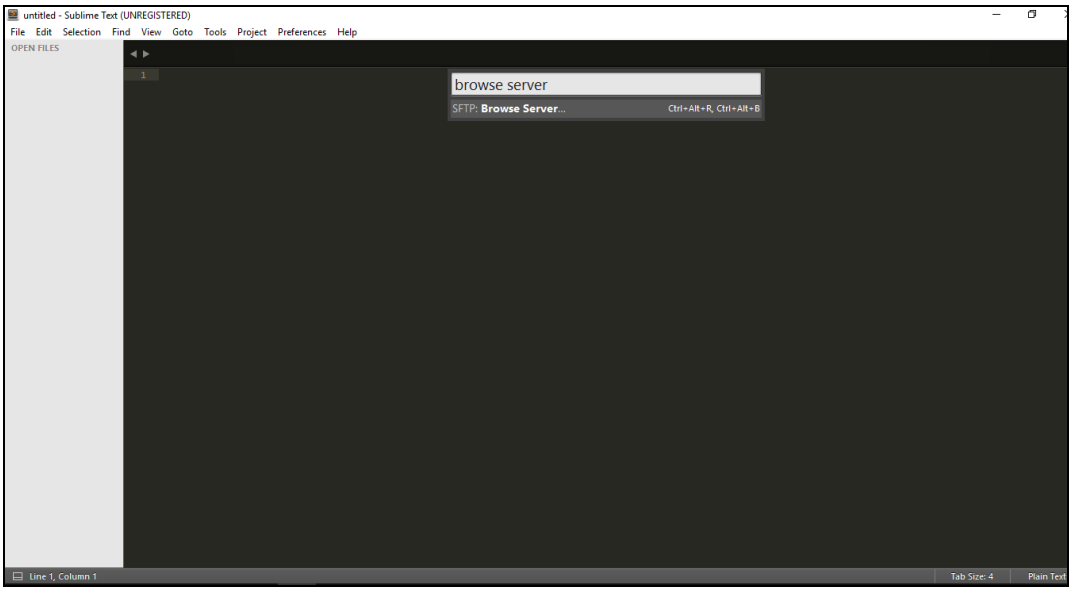


Figura 50 – Aceder aos servidores.

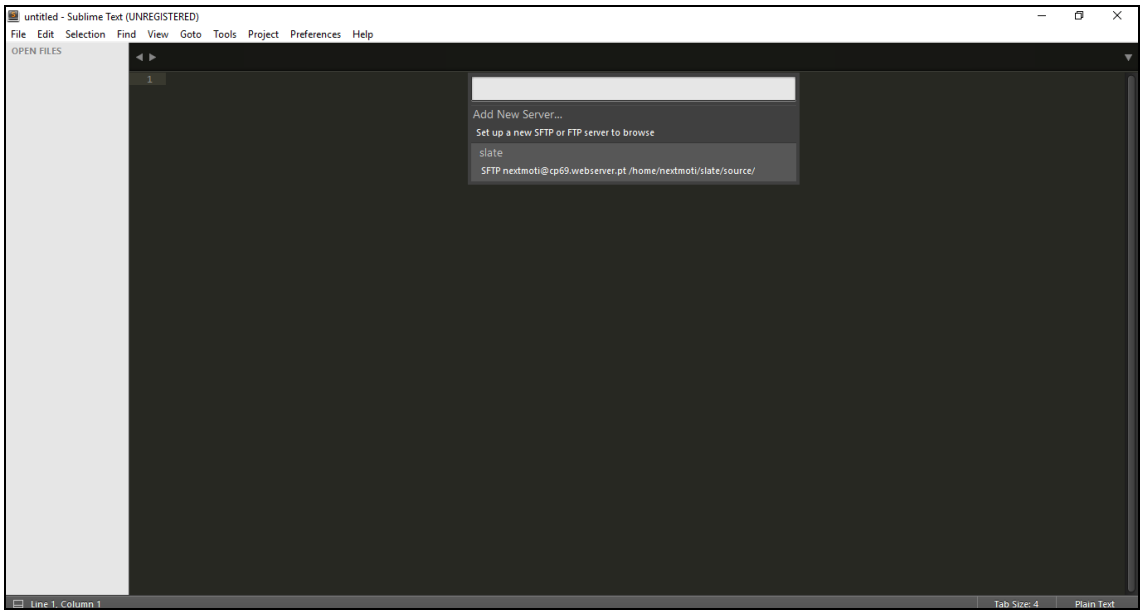


Figura 51 – Selecionar o servidor que configuramos: *Slate.*

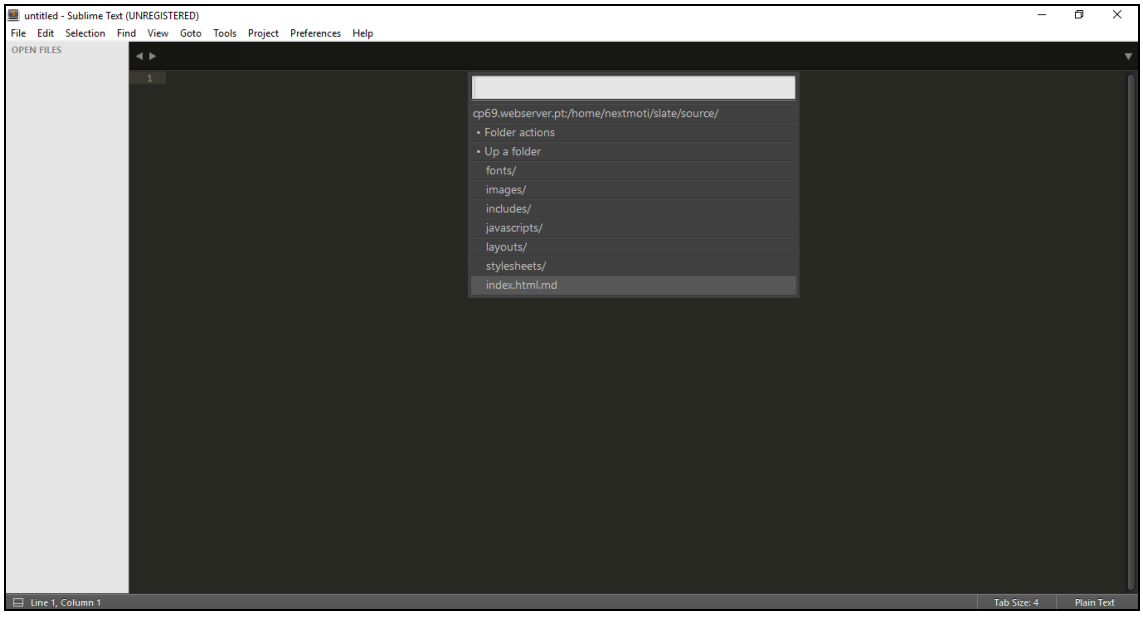


Figura 52 – Pastas que se encontram no servidor.

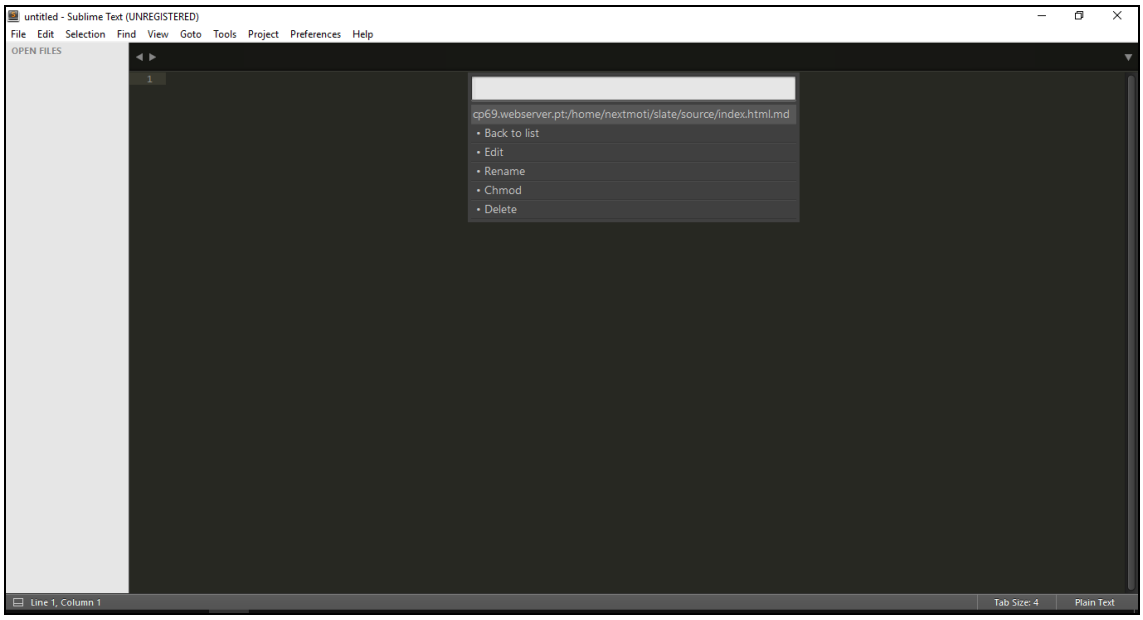


Figura 53 – Ações que podemos efetuar sobre o ficheiro selecionado.

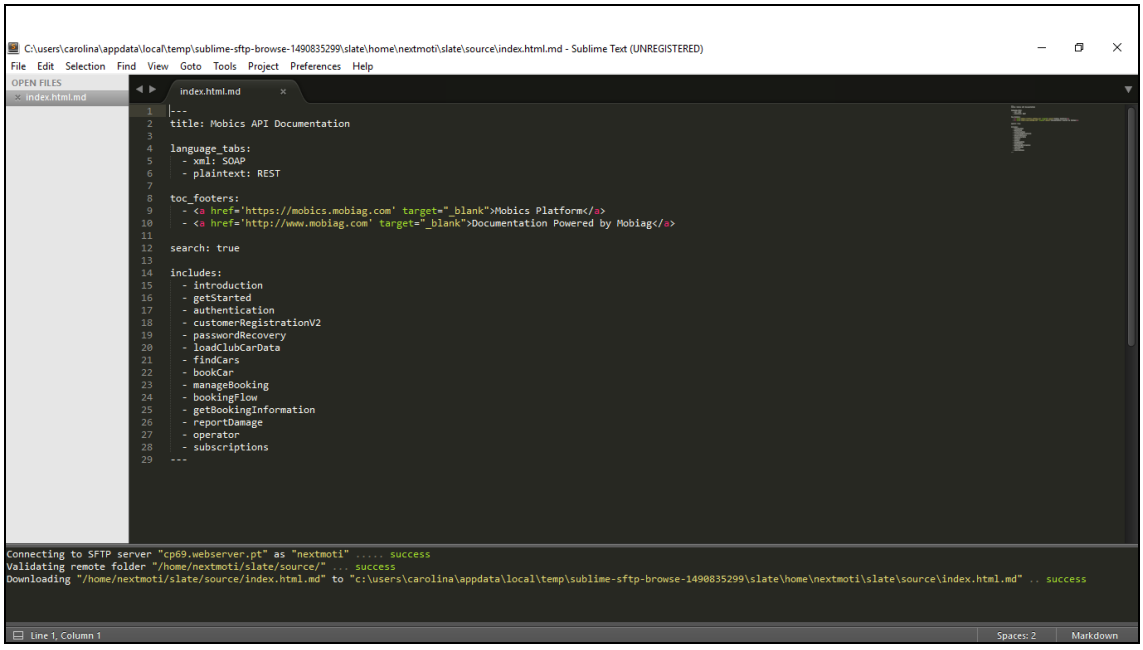


Figura 54 – Ficheiro *index.html.md* aberto e pronto a ser utilizado.

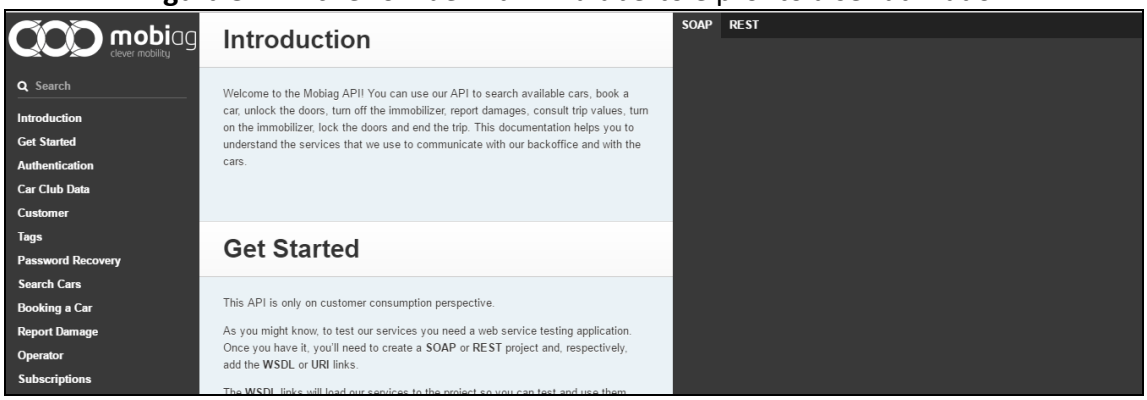
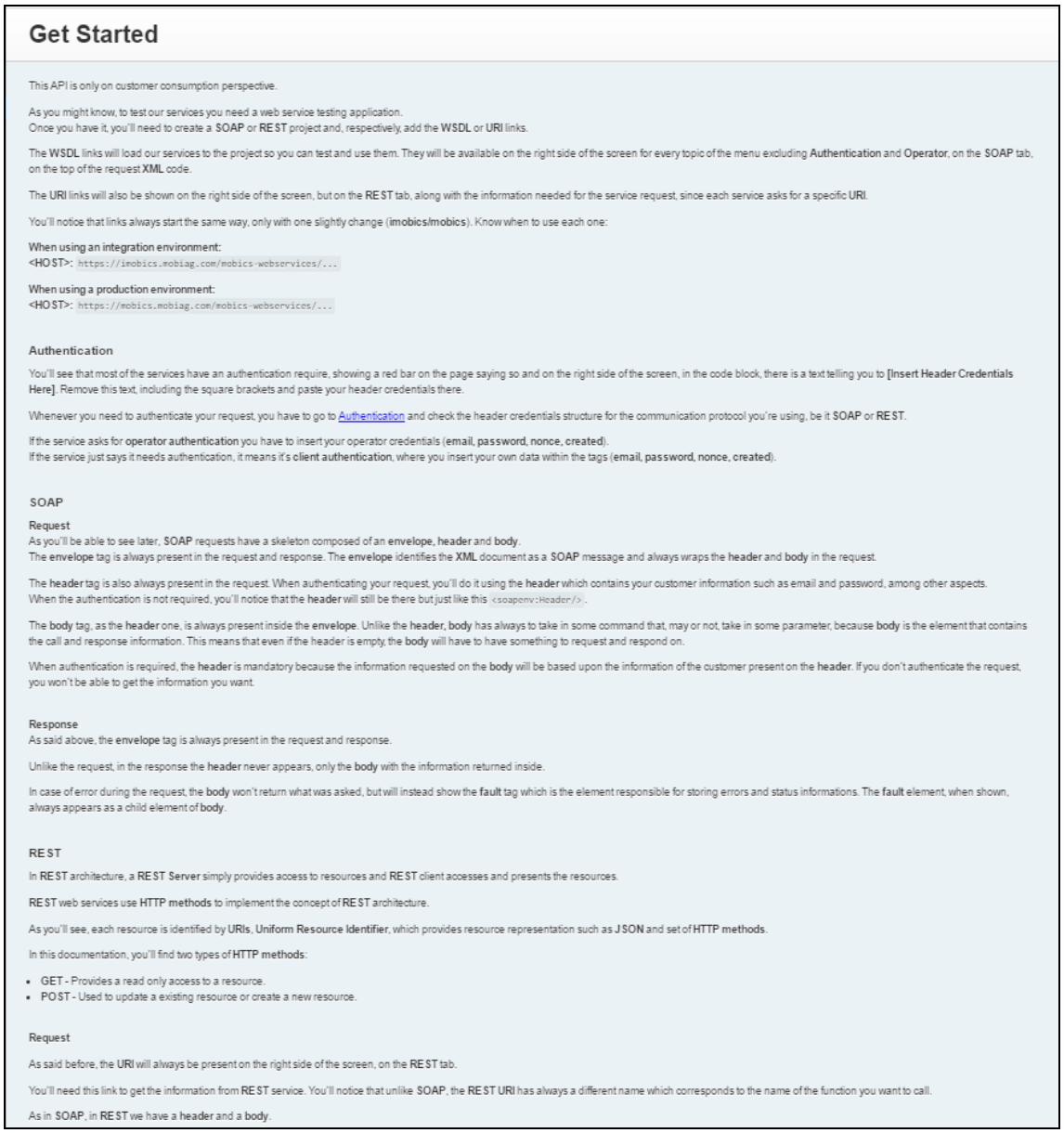


Figura 55 – Website Documentação da API Mobiag



Get Started

This API is only on customer consumption perspective.

As you might know, to test our services you need a web service testing application.
Once you have it, you'll need to create a **SOAP** or **REST** project and, respectively, add the **WSDL** or **URI** links.

The **WSDL** links will load our services to the project so you can test and use them. They will be available on the right side of the screen for every topic of the menu excluding **Authentication** and **Operator**, on the **SOAP** tab, on the top of the request **XML** code.

The **URI** links will also be shown on the right side of the screen, but on the **REST** tab, along with the information needed for the service request, since each service asks for a specific **URI**.

You'll notice that links always start the same way, only with one slightly change (**imobics/mobics**). Know when to use each one:

When using an integration environment:
<HOST>: `https://imobics.mobiag.com/mobics-webservices/...`

When using a production environment:
<HOST>: `https://mobics.mobiag.com/mobics-webservices/...`

Authentication

You'll see that most of the services have an authentication require, showing a red bar on the page saying so and on the right side of the screen, in the code block, there is a text telling you to **[Insert Header Credentials Here]**. Remove this text, including the square brackets and paste your header credentials there.

Whenever you need to authenticate your request, you have to go to Authentication and check the header credentials structure for the communication protocol you're using, be it **SOAP** or **REST**.

If the service asks for **operator authentication** you have to insert your operator credentials (**email, password, nonce, created**).
If the service just says it needs authentication, it means it's **client authentication**, where you insert your own data within the tags (**email, password, nonce, created**).

SOAP

Request
As you'll be able to see later, **SOAP** requests have a skeleton composed of an **envelope**, **header** and **body**.
The **envelope** tag is always present in the request and response. The **envelope** identifies the **XML** document as a **SOAP** message and always wraps the **header** and **body** in the request.

The **header** tag is also always present in the request. When authenticating your request, you'll do it using the **header** which contains your customer information such as email and password, among other aspects.
When the authentication is not required, you'll notice that the **header** will still be there but just like this `<soapenv:Header/>`.

The **body** tag, as the **header** one, is always present inside the **envelope**. Unlike the **header, body** has always to take in some command that, may or not, take in some parameter, because **body** is the element that contains the call and response information. This means that even if the header is empty, the **body** will have to have something to request and respond on.

When authentication is required, the **header** is mandatory because the information requested on the **body** will be based upon the information of the customer present on the **header**. If you don't authenticate the request, you won't be able to get the information you want.

Response
As said above, the **envelope** tag is always present in the request and response.

Unlike the request, in the response the **header** never appears, only the **body** with the information returned inside.

In case of error during the request, the **body** won't return what was asked, but will instead show the **fault** tag which is the element responsible for storing errors and status informations. The **fault** element, when shown, always appears as a child element of **body**.

REST

In **REST** architecture, a **REST Server** simply provides access to resources and **REST** client accesses and presents the resources.

REST web services use **HTTP methods** to implement the concept of **REST** architecture.

As you'll see, each resource is identified by **URIs**, **Uniform Resource Identifier**, which provides resource representation such as **JSON** and set of **HTTP methods**.

In this documentation, you'll find two types of **HTTP methods**:

- **GET** - Provides a read only access to a resource.
- **POST** - Used to update a existing resource or create a new resource.

Request

As said before, the **URI** will always be present on the right side of the screen, on the **REST** tab.

You'll need this link to get the information from **REST** service. You'll notice that unlike **SOAP**, the **REST URI** has always a different name which corresponds to the name of the function you want to call.

As in **SOAP**, in **REST** we have a **header** and a **body**.

Figura 56 – Introdução da documentação da API feita por mim.

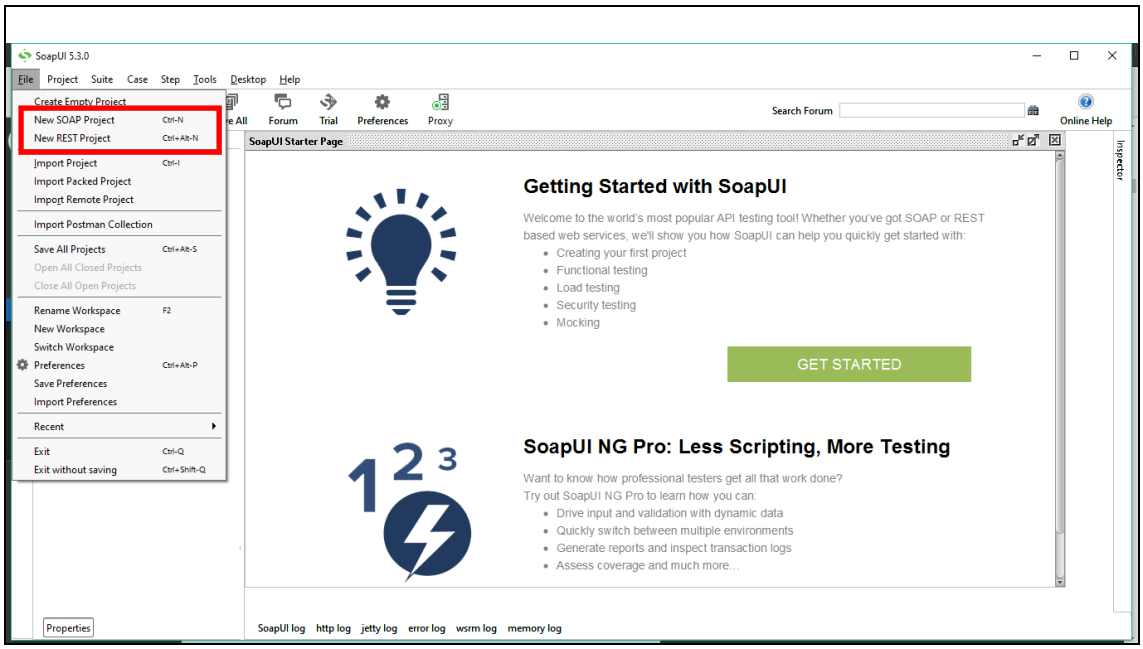


Figura 57 – Interface do SoapUI: como criar um novo projeto.

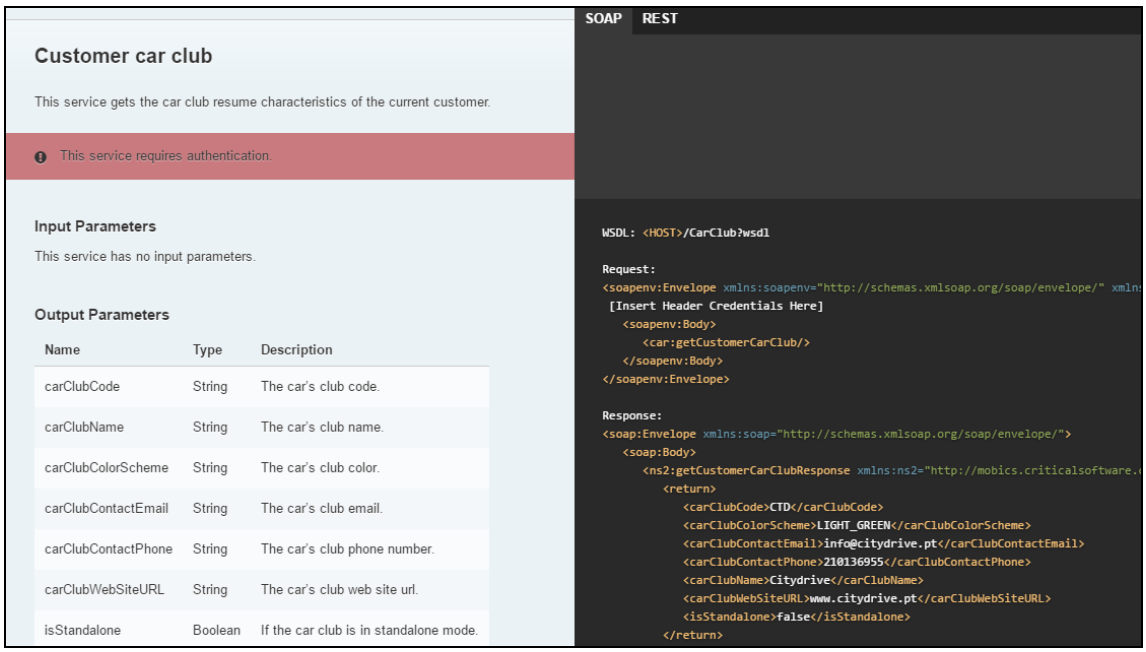


Figura 58 – Estrutura da informação na documentação.

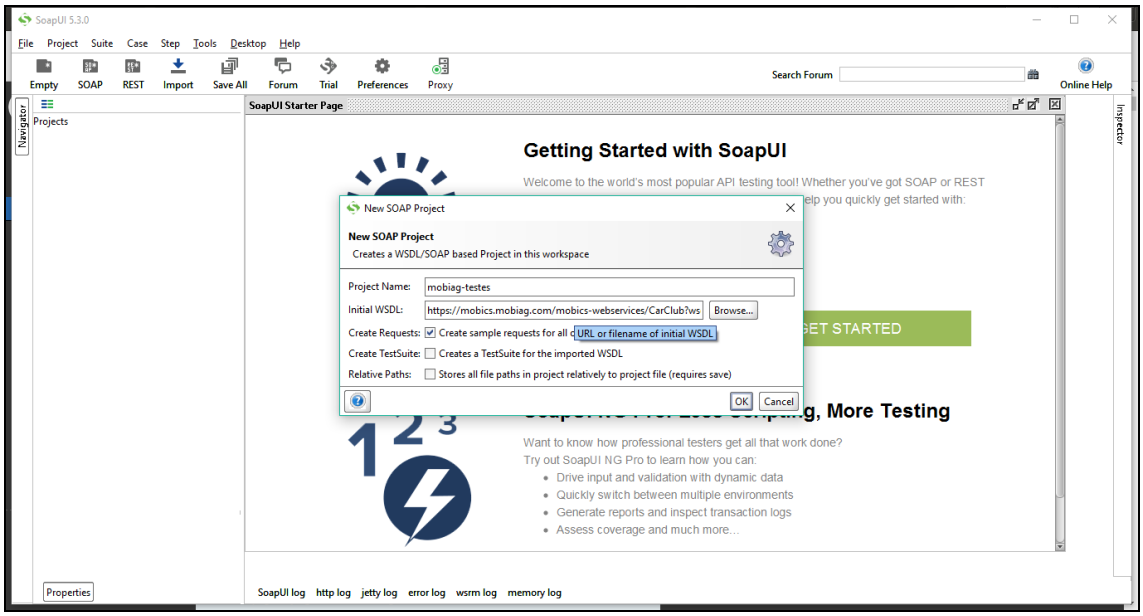


Figura 59 – Colocação do WSDL.

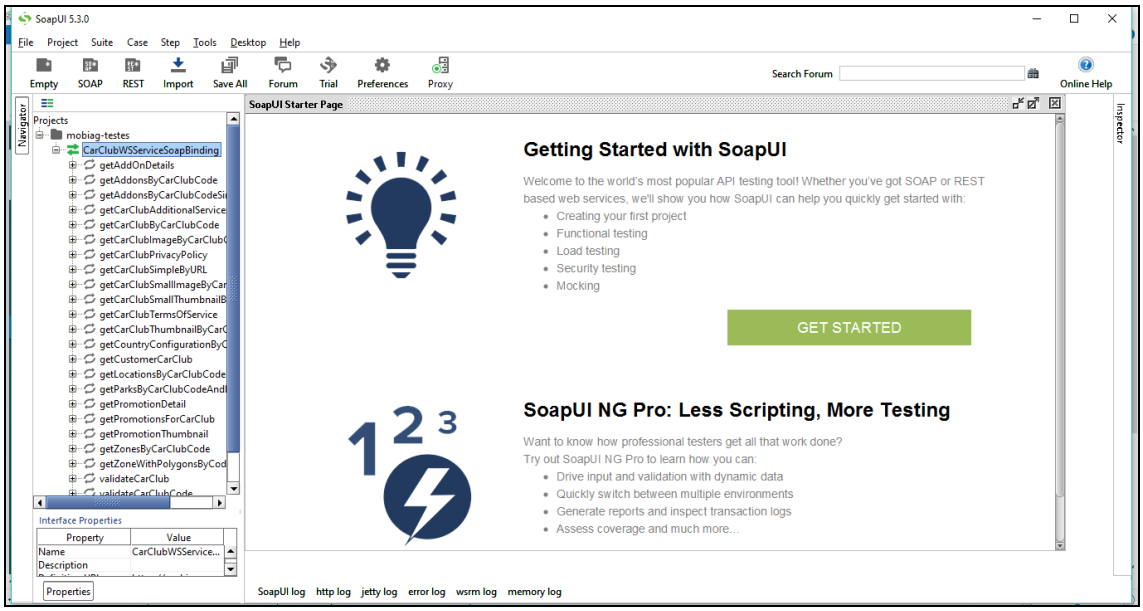


Figura 60 – SoapUI carregou todos os serviços que estão visíveis à esquerda.

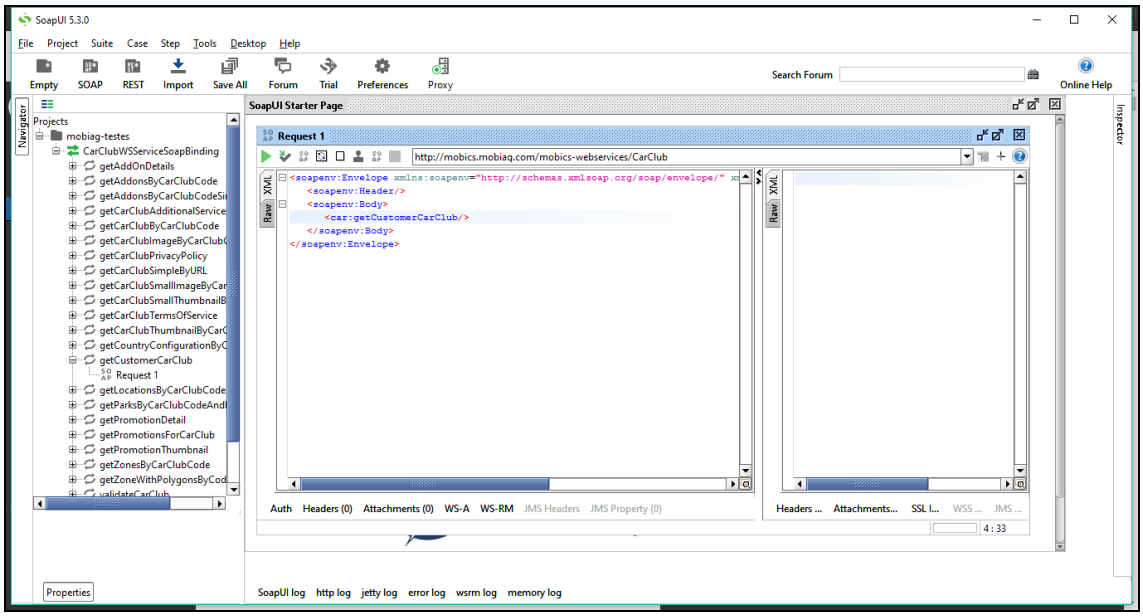


Figura 61 – Pedido com a mensagem em SOAP para o *getCustomerCarClub.*

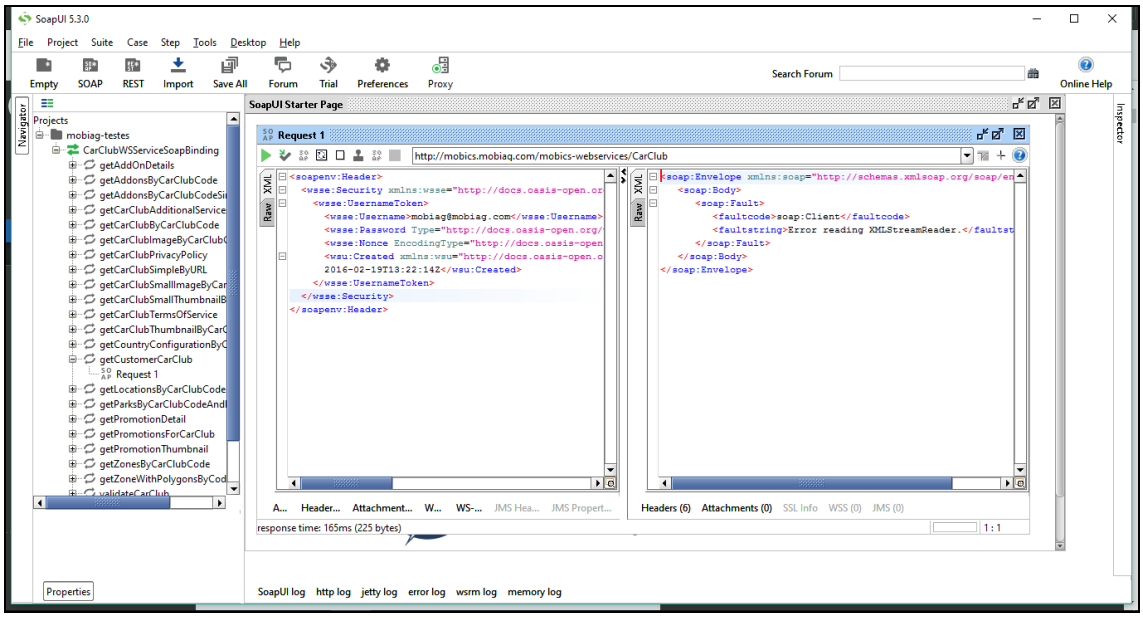


Figura 62 – Resposta em SOAP com tag *fault*, pois na altura deste printscreen, não tinha os dados de autenticação corretos para pôr no *header*.

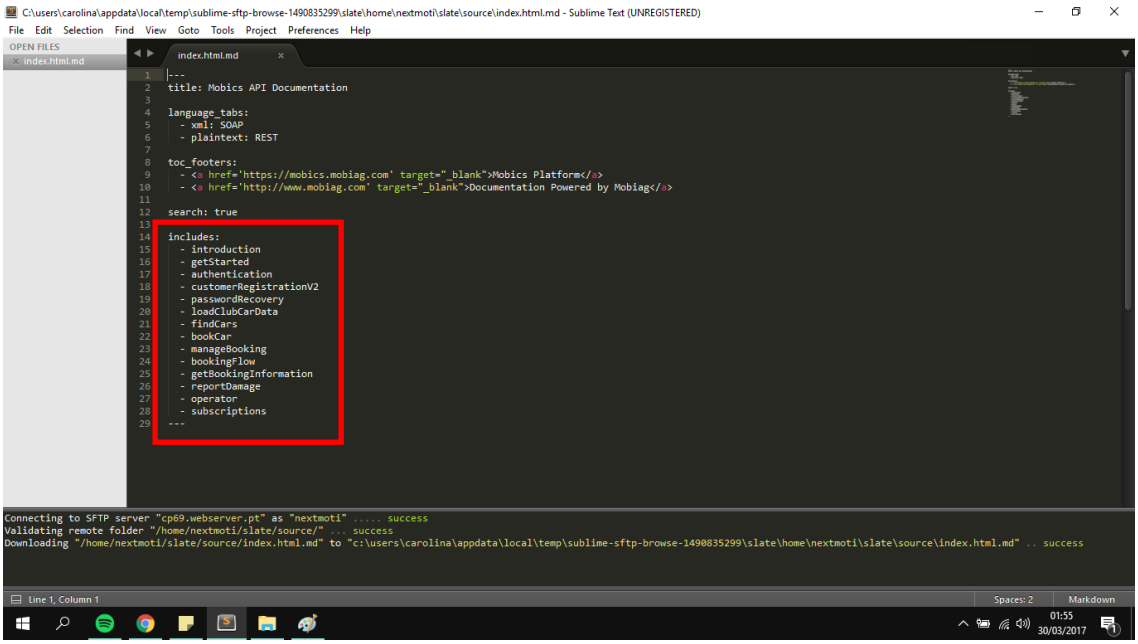


Figura 63 – Ficheiro principal, onde são chamados os *includes*.

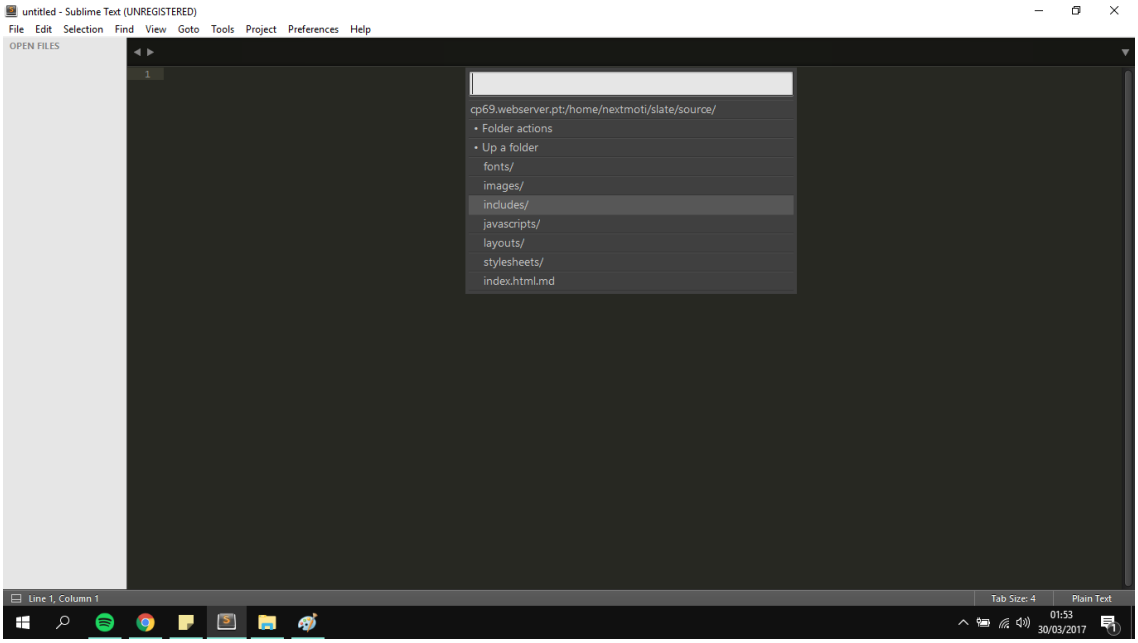


Figura 64 – Palete de comandos para aceder à pasta dos *Includes*.

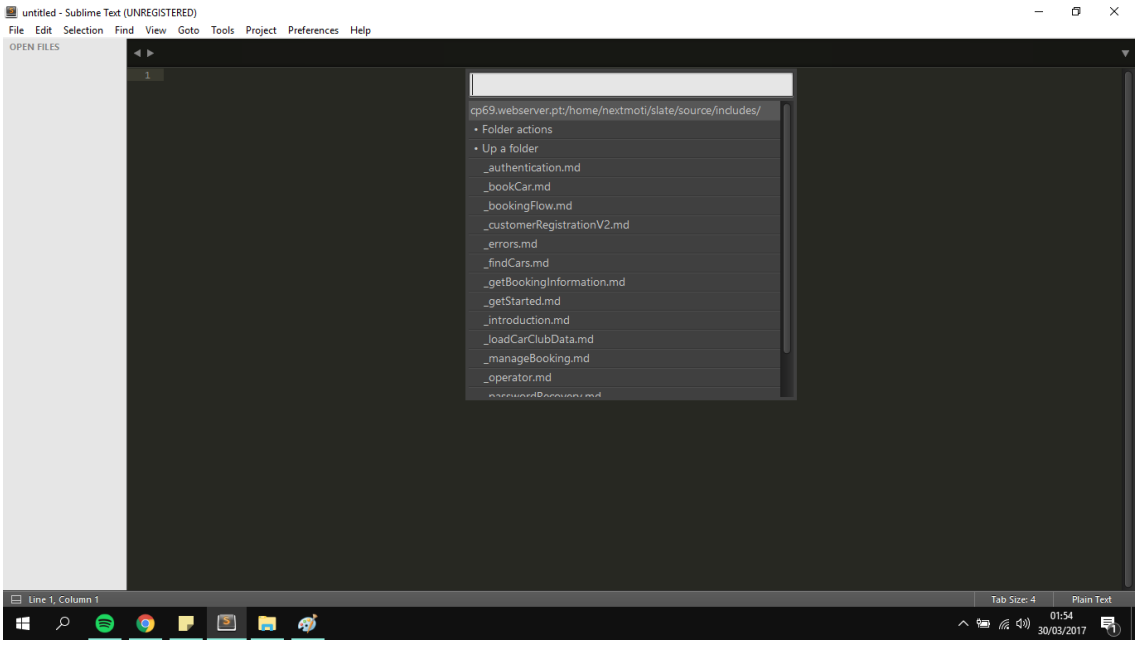


Figura 65 – Todos os *includes* disponíveis. É só selecionar o que queremos.

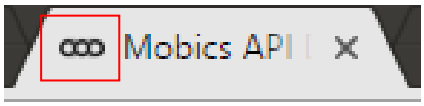


Figura 66 - Favicon